

Кафедра Конструирования и производства микроэлектронной аппаратуры

Методические указания к выполнению лабораторных работ по дисциплине СД.Ф.05 Компьютерное моделирование, расчет и проектирование наносистем

Рекомендуется УМЦКГТУ им. А.Н. Туполева для направления

направление 210600 «Нанотехнологии»

специальность 210601 «Нанотехнологии в электронике»

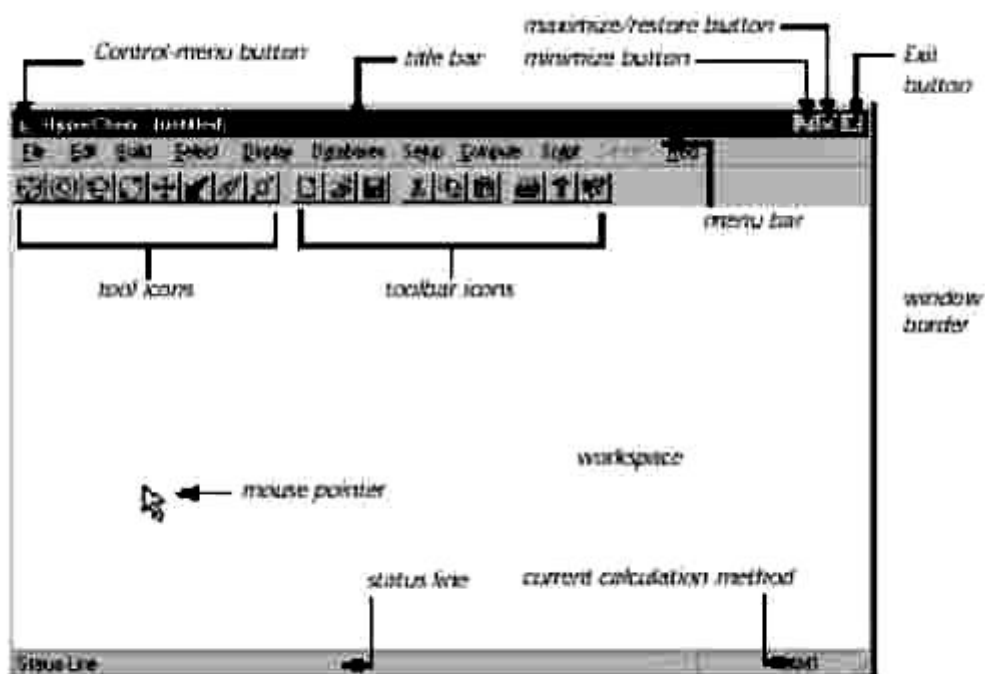
форма обучения очная

Материалы к лабораторным работам и СРС

На сегодняшний день методы квантовой химии и молекулярной динамики получили широкое распространение в численном моделировании электронной и атомной структур сложных систем молекулярных, кристаллических и переходных (нано) размеров. Это связано с технологическим развитием соответствующего математического обеспечения. Сейчас в мире функционирует достаточно много современных вычислительных комплексов, реализующих методы квантовой химии и молекулярной динамики, однако, для широкого круга пользователей наиболее доступно использование этих методов обеспечивается известной квантово-химической и молекулярно-динамической программой **HyperChem**. Все результаты молекулярно-динамического моделирования, представленные в данной мультимедийной книге получены с использованием различных версий этой программы. Бесплатную демонстрационную версию этой программы читатель может получить на сайте корпорации **Hypercube** (www.hyper.com). Весьма полезную, интересную и более доступную информацию, касающуюся версий программы, имеющих хождение в нашей стране можно также получить и по адресу www.prognauka.narod.ru. Нужно отметить, что данный интернет-ресурс существовал еще до начала написания данной книги.

Документация к программе **HyperChem** на английском языке находится в файлах **CDK.pdf**, **GetStart.pdf**, **Referenc.pdf**. Для читателей, абсолютно незнакомых с методами и программами квантовой химии и молекулярной динамики в данной Главе приводится некий “курс молодого бойца”, который позволяет легко начать работу с программой **HyperChem**.

После установки и запуска программы на мониторе появляется окно (вне зависимости от версии программы):



Строка названия показывает имя файла, с которым Вы работаете. Если Вы работаете во вновь созданном файле, имя появляется как *untitled*.

Строка меню содержит имена других **HyperChem** меню:

File Файл,
Edit Редактирование,
Build Построение объектов,
Select Выделение,
Display Отображение,
Databases Базы данных,
Setup Установки,
Compute Расчет,
Cancel Отмену,
Script Сценарий,
Help Подсказку.

Панель инструментов Левая сторона строки инструментов содержит восемь клавиш инструментальных средств, которые Вы используете, чтобы построить, выбрать, отобразить, и переместить атом или молекулу (см. ниже "Использование мыши").

Строка состояния показывает текущую информацию, такую как количество атомов в молекуле, которое к настоящему времени отображается, состояние вычисления, энергию или величину градиента. Когда Вы выбираете пункт меню, краткое описание пункта появляется в строке состояния.





Кнопки управления меню содержат команды, позволяющие менять размеры, перемещать, расширять и закрывать окно **HyperChem**. Они также содержат команду Ключа, которая позволяет Вам активизировать другие окна.

Использование мыши

Работа в **HyperChem** по большей части происходит при помощи мыши. Она используется следующим образом:

1. **Точка** - перемещение точки (скольжение), при этом курсор указывает, что Вы хотите выбрать в окне **HyperChem**.
2. **L-Нажатие** - щелчок левой кнопкой мыши.
3. **R-Нажатие** - щелчок правой кнопкой мыши. Обычно, R-нажатие имеет противоположный эффект L-нажатию.
4. **Двойное нажатие** - быстрый двойной щелчок левой кнопкой мыши.
5. **L-протяжка** или **R-протяжка** - нажатие на левую или правую кнопку мыши, и перемещение, не отпуская кнопки, (скольжение) курсора в новую позицию в рабочей области.
6. **LR-протяжка** - нажатие и удержание левой кнопки мыши, затем правой кнопки и перемещение курсора в новую позицию в рабочей области.
7. **RL-протяжка** - так же, как LR-протяжка, но вначале нажатие правой кнопки мыши.

Курсор мыши способен изменять форму. Чтобы это увидеть:

1. Укажите на инструментальное изображение (**Draw**) нажмите левую клавишу .
2. Переместите курсор в рабочую область. Курсор приобретает соответствующую форму: .
3. Укажите на инструментальное изображение (**Select**) и нажмите левую клавишу .
4. Переместите курсор в рабочую область. Он приобретает соответствующую форму: . Таким же образом изменяется форма курсора при выборе остальных шести клавиш строки инструментов.



Клавиатурные альтернативы

HyperChem обеспечивает стандартные альтернативы клавиатуры

Чтобы открыть меню, использующее клавиатурную альтернативу, нажмите одновременно кнопку **[Alt]** и клавишу **[S]**. Открывается меню *Выбора (Select)*.

Таким образом, можно открыть и все остальные меню, нажимая одновременно с **[Alt]** на клавишу **первой буквы названия** (F для меню File и т.д.).

Чтобы закрыть меню: нажмите **[Alt]** или клавишу **[Esc]**.

Чтобы выбрать один из пунктов уже открытого меню: нажмите одновременно на кнопку **[Alt]** и клавишу, соответствующую **первой букве названия пункта меню** (A для пункта Atom).

Кратчайшие клавиатурные пути

HyperChem имеет различные кратчайшие клавиатурные пути.

[Ctrl] + [N] - создать новый файл (*New* на Файловом меню).

[Ctrl] + [O] - открыть файл (*Open* на Файловом меню).

[Ctrl] + [S] - сохранить (*Save*)

[Ctrl] + [X] - вырезать (*Cut*)

[Ctrl] + [C] - копировать (*Copy*)

[Ctrl] + [V] - вставить (*Paste*)

[Alt] + [F4] - выход (*Exit*)

[F4] - построение изоповерхностей

[F9] - копирование всего изображения (*Copy ../book/image(III)/Image*) в меню

Редактирования.

[Esc] - отмена.

Открытие файла образца *Sample File*

В программе **HyperChem** Вы можете работать с молекулами тремя способами:

1. Инструментальные средства дают возможность создать двухмерную (**2D**) структуру молекулы и затем преобразовывает ее с помощью моделестроителя **HyperChem** в трехмерный (**3D**) вид.
2. Выбор остатков позволяет последовательно выбрать из **HyperChem/Lite** уже готовые остатки аминокислот и нуклеотидов (нуклеозидов), чтобы создать белки и нуклеиновые кислоты.
3. Чтение: набор атомных и молекулярных координат, которые были сохранены в **HyperChem** формате входного файла (файл **HIN**) или формате Белкового банка данных **Brookhaven Protein Data Bank** (файл **ENT**), а также в ряде других форматов.

Открытие **HIN** файла

1. Переместить указатель мыши в верхнюю левую часть окна на меню **File**.
2. Нажатием левой кнопки мыши (**L-Нажатие**) откройте файловое меню
3. В диалоговом блоке меню выберите *Open (Открыть)*

4. Появляется список файлов, среди которых необходимо выбрать нужный и **L-Нажатием** кнопки мыши открыть его.



Использование дисплейных установочных параметров

HyperChem автоматически использует дисплейные установочные параметры с последнего сеанса работы. Выбрать дисплейные установочные параметры можно, используя пункт меню *Display (Отображение)*.

Использование подписей атомов и молекул

Если молекула отображается с подписями атомных зарядов и пр., удалить их можно следующим образом:

1. **L-Нажатием** кнопки мыши открыть *Labels (Этикетки)*.
2. В списке подписей атомов автоматически помечено *None (Ничто)*.
3. **L-Нажатие** на **ОК** отменяет подписи атомов. Диалоговый ящик закрывается, и подписи не отображаются.

Чтобы использовать подписи необходимо в диалоговом окне пометить интересное и нажать **ОК**. Диалоговый ящик закрывается, и молекула помечается выбранными символами (химическими символами элементов, зарядами и пр.).

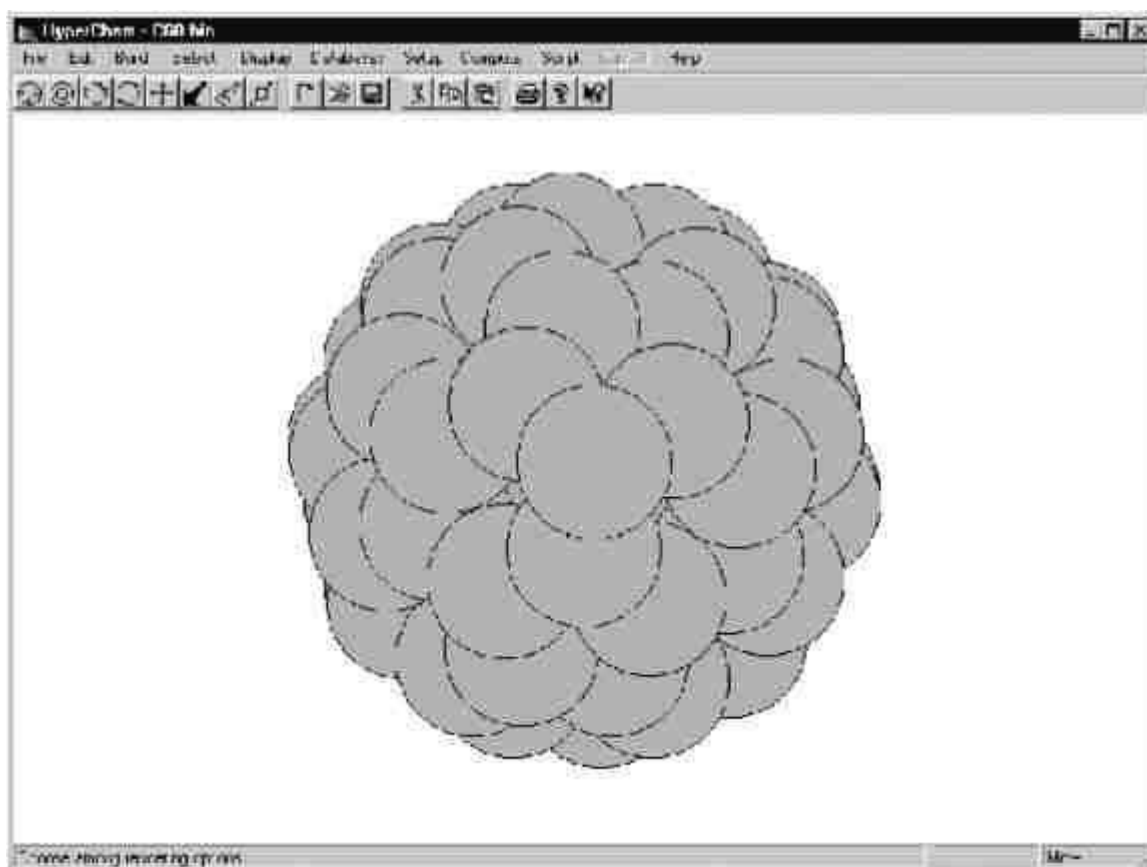
Использование молекулярных изображений

Чтобы изменить изображения молекулярной системы:

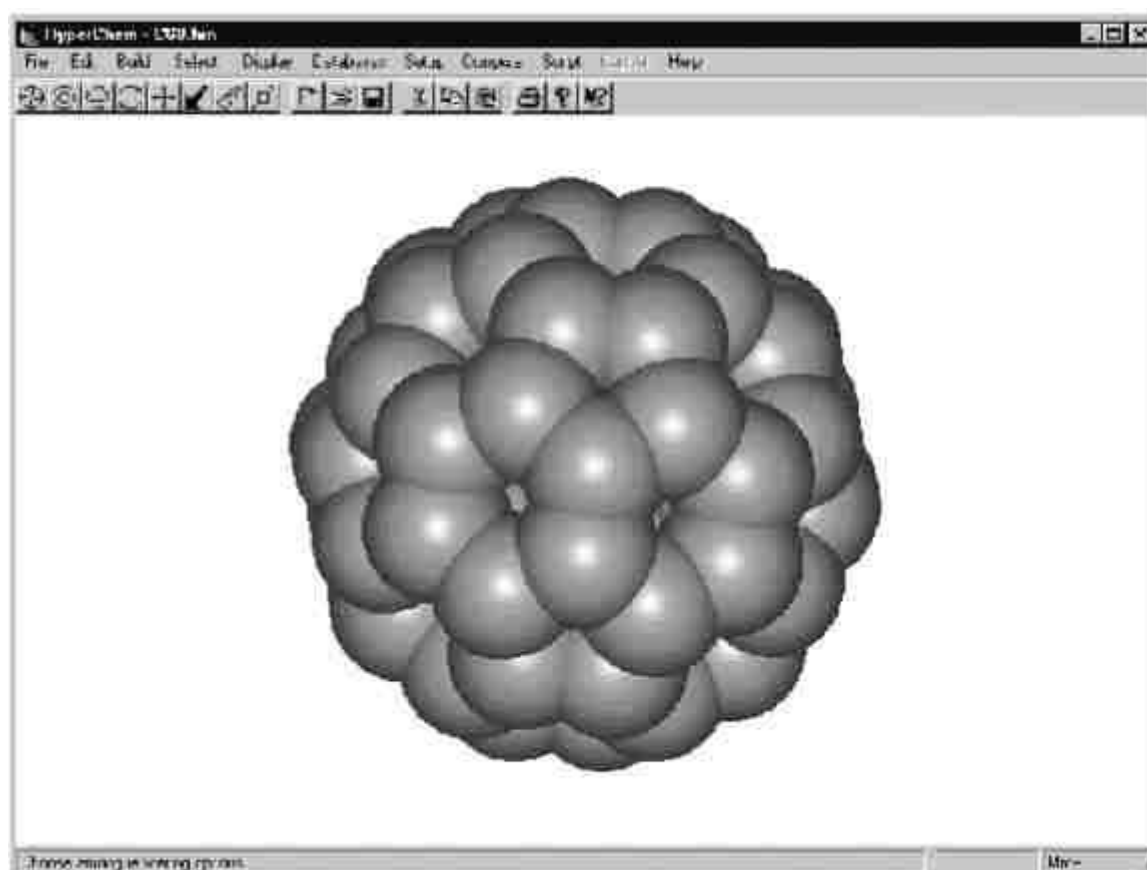
1. Выбрать *Renderings* в *Display* меню.
2. В диалоговом меню выбрать и пометить, например, *Bolls (Шары)*.
3. В верхней части диалогового окна нажать на клавишу *Bolls*, которая открывает лист параметров



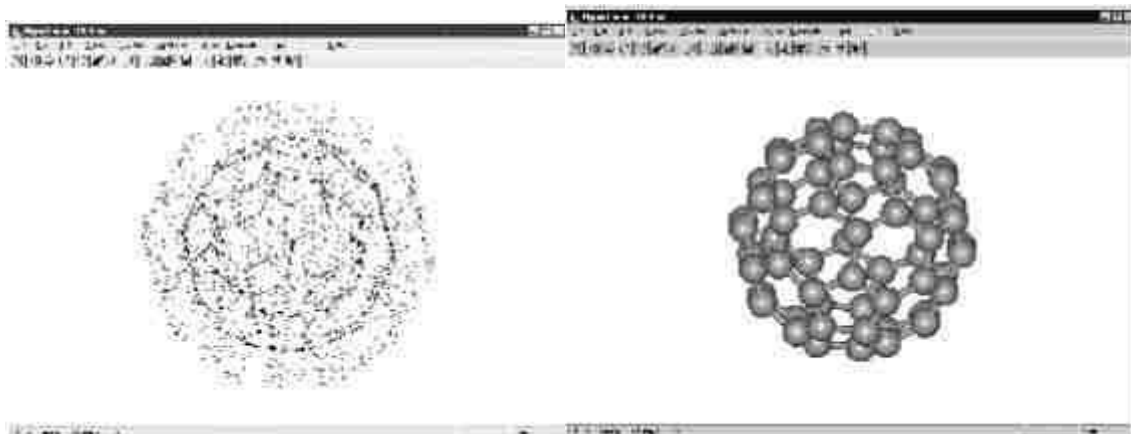
4. Если удалить метку (✓) *Shading* (*Затенения*) и нажать **ОК**, будет получено приближенное, но быстро обрабатываемое объемное изображение молекулы.



5. Если же отметить *Shading* и затем нажать **OK**, это даст пространственно заполненное изображение.



6. Если в диалоговом меню выбрать \vee *Sticks & Dots* (*Стержни и точки*). Такое представление хорошо демонстрирует форму молекулы.



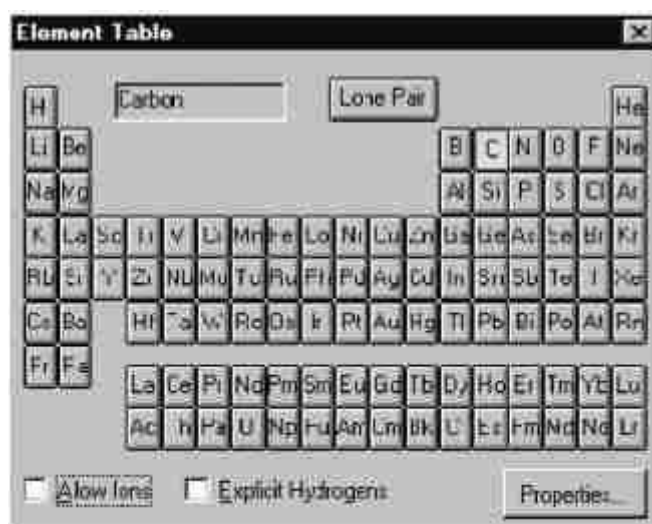
Нажатие клавиши [F2] позволяет восстановить параметры изображения молекулы, выбранные ранее. Это - эквивалент выбора *Last Rendering* (*Последний раз*) в дисплейном меню.

Основная техника построения и редактирования объектов

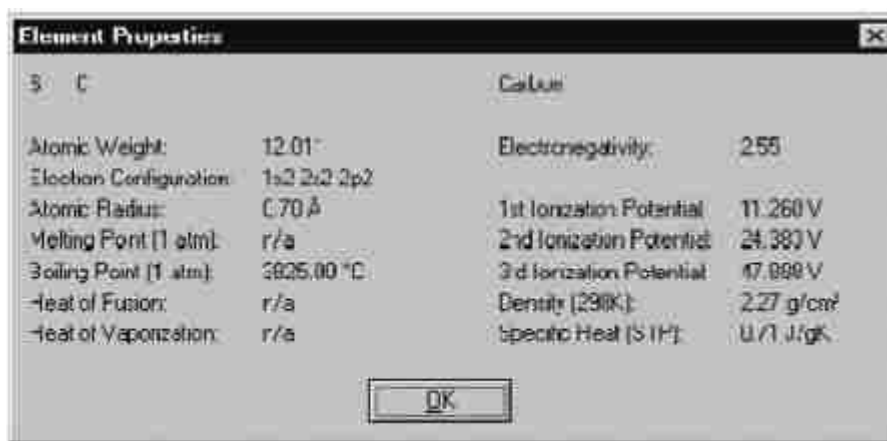
После открытия *HyperChem* желательно расширить окно на полный экран. В большой рабочей области гораздо удобнее строить молекулы.

Рисование отдельных атомов:

1. Откройте в меню *Build* пункт *Default element*. В нем содержится диалоговое меню *Element Table* - периодическая таблица элементов Д. И. Менделеева.



2. Для выбора элемента курсор мыши наведите на желаемый элемент и щелкните левой клавишей.
3. Если нажать на кнопку *Properties...*, откроется окно, содержащее информацию о физических свойствах выбранного элемента.



После нажатия ОК окно исчезает.

4. Можно также отметить ☐ *Allow ions* (Допустимы ионы) или *Explicit hydrogens* (Добавить водороды).

5. После выбора элемента закройте *Element Table*.

6. Переместите курсор в рабочую область. При этом он приобретает форму прицела Å

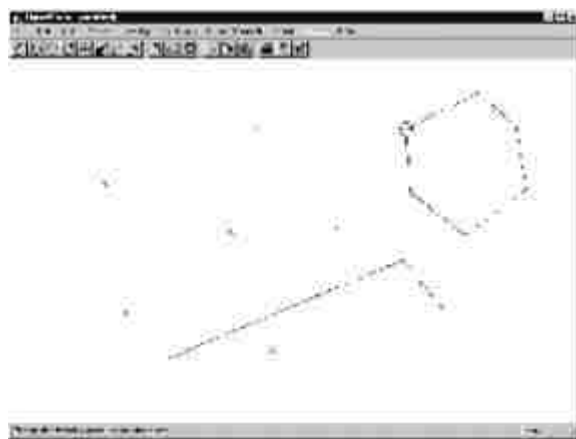
7. Разместите в поле построения выбранный атом, нажав однократно на левую кнопку мыши. Так можно разместить на любом расстоянии друг от друга неограниченное число атомов одного элемента.


8. Для смены элемента вновь необходимо открыть *Default Element* и щелкнуть по нему левой кнопкой мыши.



9. Для соединения двух атомов между собой "прицел" мыши Å (**Draw**) наведите на атом и, не отпуская левой клавиши (**Л-протяжка**), проведите линию до другого атома. Таким образом, все размещенные в пространстве атомы можно соединить в одну молекулу посредством химических связей.

10. Существует другой способ построения молекулы. После того, как Вы разместили один атом в поле построения, не отпуская левой клавиши (**Л-протяжка**) проведите от него линию до места, где должен находиться второй атом и, не отпуская кнопки мыши, однократно щелкните. Это позиция второго атома.



11. При построении молекул необходимо учитывать, что не все из них имеют плоскую структуру, поэтому для размещения отдельных атомов в пространстве под определенным углом относительно друг друга молекулу можно разворачивать, выбрав указатель мыши  *Rotate out of plane*.
12. Чтобы химическую связь сделать кратной (двойной, тройной, полуторной), прицел \AA необходимо разместить рядом с линией, изображающей одинарную связь и однократно (двукратно) нажать на левую клавишу мыши.
13. Для удаления связей или атомов пользуются правой клавишей мыши (**R-щелчок**).

Работа с выделенными атомами (молекулами)

Удалить или копировать сразу несколько атомов или всю молекулу можно после их выделения. Для этой цели служит кнопка \AA (вложенные друг в друга пара кружочков).

1. Вначале необходимо выбрать параметры выделения в меню *Select*. Можно выделять как отдельные *атомы*, *остатки (residues)*, так и целые *молекулы*, по отдельности или вместе.
2. После того, как Вы настроили параметры выделения (например, \AA атом), нужно привести курсор на выделяемый объект в рабочем поле и сделать **L-щелчок** кнопкой мыши.
3. Таким же образом можно выделить связь между двумя атомами. В этом случае в параметрах выделения должен быть обязательно отмечен \AA *Atoms*.

Выделенные фрагменты можно удалить или запомнить в буфер с последующим извлечением из него (соответствующие кнопки на меню или разделы в меню *Edit*).

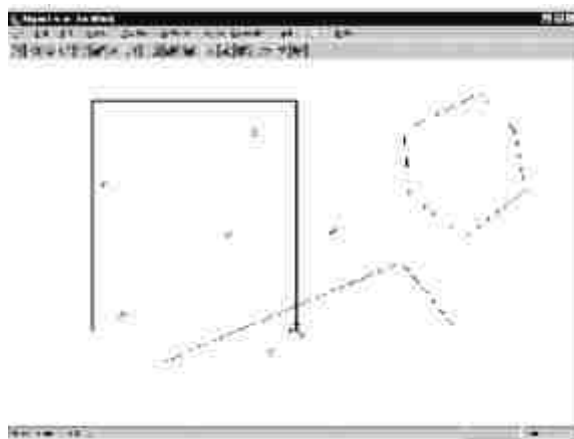
Выделенные фрагменты можно также отдельно перемещать или вращать на рабочем поле, выбрав соответствующие инструменты и “схватив” их правой кнопкой мыши (только для этого необходимо в меню *File/Preferences/Tool* отменить (\AA) *whole molecular translation*).

Для отмены выделения поместите курсор в пустой области экрана и щелкните левой кнопкой мыши. Чтобы отменить выделение отдельного атома, фрагмента или связи курсор \AA наводят на выделенный объект и однократно щелкают правой клавишей мыши.

Выбор группы атомов

При выборе группы атомов они заключаются в прямоугольник

1. В меню *Select* не должно быть помечено *Select Sphere*.
2. Выберите точку в пустой части рабочей области, проведите от нее линию, одновременно удерживая левую и правую клавиши мыши (**LR -протяжка**) к выделяемым атомам. При этом будет рисоваться прямоугольник, отображающий границу области выбора.
3. Отпустите кнопки мыши, и все атомы области выбора окажутся выделенными. В этом же окне можно выделить вторую группу атомов. Для этого в меню *Select* необходимо отметить (\AA) *Multiple Selections*. После чего таким же образом в прямоугольник заключить другую группу атомов. При этом новые выделенные атомы добавятся к предыдущим.



Чтобы выделить все атомы нужно в пустом пространстве рабочей области поместить курсор **☒** и однократно щелкнуть левой кнопкой мыши. Для отмены всех выделений нужно однократно нажать на правую кнопку мыши, когда курсор находится в пустом пространстве рабочей области.

Удаление атомов

Для удаления одного атома или связи:

1. Произведите однократный щелчок на инструментальное изображение **A (Draw)** панели инструментов.
2. Наведите курсор на удаляемый объект и произведите **R-щелчок**. Атом или связь исчезнут

Для удаления нескольких атомов или связей:

Посредством **LR-протяжки** заключите в пространство внутри прямоугольника атомы и связи, которые подлежат удалению

В меню *Edit (Редактирования)* выберите *Clear (Очистить)*. Появится диалоговый блок, где будет задан вопрос: "Хотите ли вы удалить выбор?". Нужно нажать "Да".

Копирование атома в буфер:

1. В меню *Select* отметьте *☑ Atoms*.
2. Щелчком левой клавиши в рабочем поле выделить атом или связь.
3. В меню *Edit (Редактирования)* выбрать *Copy (Копирование)*. Копия атома или связи сохраняется в буфере.
4. Для вставки объектов из буфера в рабочую область можно выбрать в меню редактирования команду *Past (Вставить)*.

Очистка рабочей области HyperChem:

Выберите в меню *File (Файл)* пункт *New (Новый)*. Появляется диалоговое окно, в котором содержится вопрос: "Хотите ли вы сохранить текущие изменения в данном файле?".

Нужно выбрать "No". **HyperChem** очищает рабочую область.

Создание полипептидов

До сих пор Вы учились строить отдельные молекулы и отображать их, читая координаты из файла **HIN**. В этом разделе изложены принципы построения полипептидов посредством последовательного выбора остатков аминокислот из библиотеки **HyperChem**.

Чтобы открыть диалоговое меню библиотеки аминокислот:

Выберите меню *Databases* (База Данных) пункт *Amino Acids* (Аминокислоты).

Это диалоговое меню является устойчивым и остается открытым все время, пока Вы строите полипептид.



Последовательно выбирая остатки, Вы строите вторичную структуру полипептида. Но для этого нужно в диалоговом окне отметить, что должна представлять из себя эта структура: *альфа-спираль* (*Alpha helix*), *бета-лист* (складчатость) (*Beta sheet*) или другие варианты. Автоматически устанавливаются как *phi* так и угол *psi*. Угол омеги (*Omega*) можно изменить но обычно это 180 ° для транс-пептидной связи. Начинайте построение с N-конца полипептидной цепи.

Чтобы построить цепь:

L-щелчком последовательно выбирайте аминокислоты начиная с N-концевого остатка.

HyperChem строит цепь, располагая аминокислоты под соответствующими углами относительно друг друга

Создание цвиттер-иона

В построенной Вами полипептидной цепи N-конец содержит HN-, а C-конец -CO группу.

Создание цвиттериона модифицирует N- и C- концевые остатки аминокислот.

Чтобы создать цвиттерион в меню База данных выберите пункт *Цвиттерион*

(*Zwitterion*). **HyperChem** добавит атом кислорода на C-конец полипептида (получится COO-) и два протона к N-концу (до NH₃).

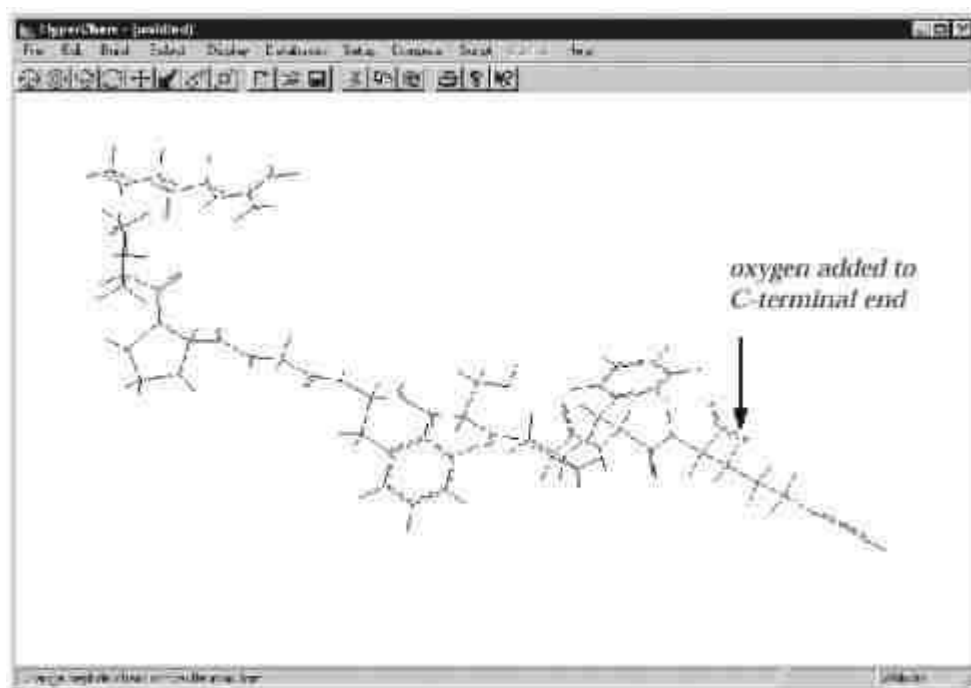
Мутагенез

Сайт-специфический мутагенез играет важную роль в белковой инженерии. Замена конкретной аминокислоты на критическом месте может изменить структуру и свойства белка, а следовательно, функцию.

Чтобы заменить остаток:

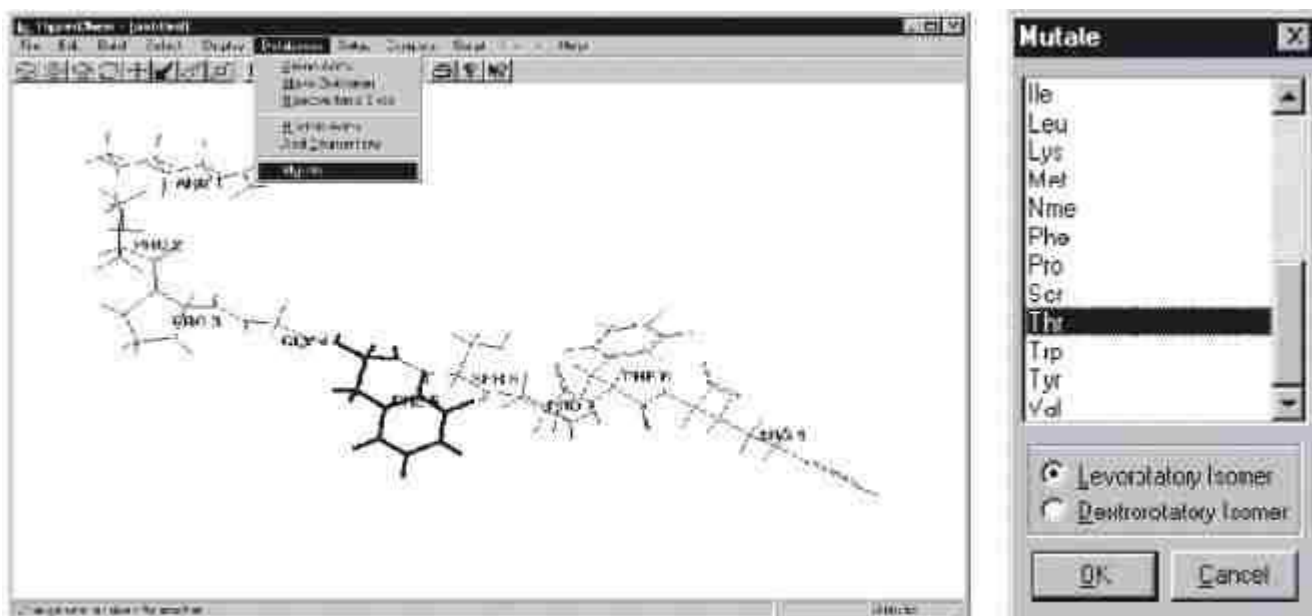
Сначала выберите аминокислоту, которую нужно заменить. Для этого в меню *Дисплей* в пункте *Этикетки* (*Labels*) отметьте *Name+Seq* как опции для маркирования остатков и нажмите ОК. Можно также в меню *Выбор* (*Select*) отметить *Остатки* (*Residues*).

После чего измените форму курсора на ¥ *Select* и L-щелчком кнопки мыши выберите нужную аминокислоту.



В меню База данных становится активным пункт *Мутировать* (*Mutate*), который ранее был неактивен

В диалоговом меню *Mutate* выберите из списка аминокислоту, на которую будет заменен выделенный остаток и нажмите ОК. Происходит замена
Сохраните полученную структуру.



Упражнения

1. Постройте следующую полипептидную цепь в бетте-конформации:
2. Создайте нуклеиновую кислоту, использующую Nucleic Acids в меню Баз Данных.

Измерение параметров структур (Measuring Structural Properties)

Данный раздел описывает технику измерения структурных связей. Дается также описание способов измерения углов, а также дисплейных характеристик атомов, как, например, заряды и пространственные координаты.

Характеристики атомов

Для того чтобы получить информацию об основных характеристиках атома его нужно выделить. В строке состояния появляются: номер атом, тип, и заряд для выбранного силового поля молекулярной механики. Она также показывает x, y и z-координаты этого атома.

Пункты меню *Build* позволяют установить необходимый Вам *Set Atom Type* (тип атома), *Set Charge* (заряд), и *Constrain Geometry* (ограниченную геометрию), отличные от тех, которые устанавливаются программой по умолчанию.

Измерение длины связи

Если Вы выбираете связь, а не атом, информация о ней появляется в строке состояния.

HyperChem имеет библиотеку длин связей между атомами конкретного типа и гибридизации, что устанавливается по умолчанию.

Когда информация о длине связи в библиотеке отсутствует, то **HyperChem** использует среднее значение ковалентных радиусов двух атомов.

Для измерения расстояния нужно изменить форму курсора на **¥ (Select)** и выделить эту связь. В строке состояния появляется значение длины связи между двумя атомами, выраженное в ангстремах (Å).

При этом в меню *Build* становится активным пункт *Constrain bond length*, который позволяет Вам установить желаемую длину связи, отличную от той, которую устанавливает Разработчик моделей программы по умолчанию.

Измерение углов связей

Для того чтобы измерить угол между двумя связями нужно последовательно выделить первый, второй и третий атомы, связи между которыми и образуют угол. При этом второй атом должен находиться в вершине этого угла. Величина угла появится в строке состояния.

В меню *Build* становится активным пункт *Constrain Bond Angle*, что позволяет Вам изменить величину данного угла.

Измерение торсионных углов

Последовательно выделите первый, второй (вершина угла) атомы и третий атом, который расположен вне плоскости молекулы. В строке состояния появится величина торсионного угла между плоскостями, в которых лежат два первых атома, и плоскостью третьего атома.

Пункт *Constrain Bond Torsion* в меню *Build* становится активным, что позволяет Вам изменить по желанию эту величину.

Измерение расстояния между двумя несвязанными атомами

Прежде, чем Вы приступите к измерению, нужно в меню *Select* отметить \checkmark пункт *Multiple Selections* (Множественные выборы). Это позволяет выделить более одного атома в рабочем окне. **Л-щелчком** \checkmark выделите два любых атома, при этом строка состояния показывает расстояние между ними.

Водородные связи

Чтобы подтвердить благоприятные условия для образования водородных связей, *HyperChem* вычисляет их и выводит на дисплей.

Водородные связи формируются, если расстояние до водородного донора - менее чем 3.2 Å и угол ковалентной связи донора и акцептора - менее чем 120 градусов.

Чтобы подтвердить условия для водородной связи:

1. В меню *Display* отметьте \checkmark пункт *Show Hydrogen Bonds* (Показать водородные связи)
2. Там же выберите *Recompute H Bonds* (Вычислить заново водородные связи)

HyperChem отображает водородные связи пунктирной линией.

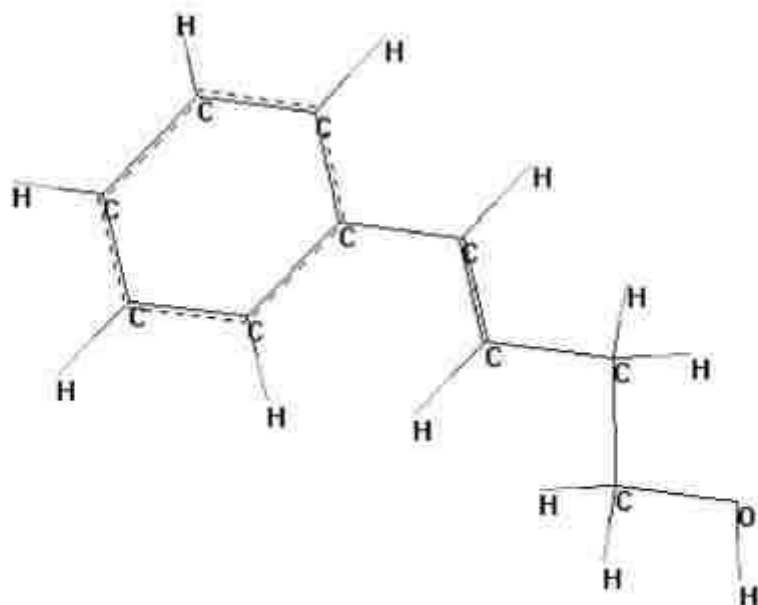
Водородные связи не вычисляются автоматически в каждой конфигурации, поэтому, когда Вы изменяете геометрию молекулы, водородные связи приходится вычислять заново.

Создание небольших молекул в 2D и 3D (Упражнение)

Теперь, когда Вы научились строить атомы и связи между ними можно приступать к построению молекулы. Хотя *HyperChem* позволяет Вам делать молекулы любого размера, по практическим соображениям мы рекомендуем Вам ограничиться построением небольших и средних молекул.

В этом упражнении, Вы будите строить молекулу 1-гидрокси-3-фенил-2 пропена.

1. Откройте в меню **Build** в **Default elements** периодическую таблицу элементов
2. Отметьте (\checkmark) **Allow ions** (Допустимы ионы) и **Explicit hydrogens** (Добавить водороды). Если **Explicit hydrogens** не выключить, то они в процессе рисования не будут автоматически добавляться к углеродному остову.
3. Выберите *углерод* (C) и закройте диалоговый ящик. Углерод устанавливается как встроенный элемент для построения
4. Теперь нарисуйте следующую структуру:



У Вас при построении могла получиться неверная геометрия, но прежде чем модифицировали эту структуру, сохраните вашу работу. Этот путь позволит Вам не рисовать все заново, а лишь вносить необходимые исправления в первоначальный вариант построенной молекулы.

Чтобы сохранить Вашу работу:

1. Выберите в файловом меню **Save ... (Сохранить)**.
2. Появляется диалоговое окно сохранения файла. Убедитесь, что файл будет сохранен в нужную папку. В **File name** введите желаемое имя файла (лучше, чтобы оно наиболее полно отражало содержание файла и параметры расчета)
3. Убедитесь, что **Save as type (Тип файла)** это **HIN**.
4. **L**-щелчком выберите ОК.
5. Диалоговое окно закрывается и в верхнем левом углу экрана над панелью инструментов появляется название файла

Теперь Вы можете модифицировать структуру, и все изменения будут сохранены в этом файле (для сохранения можно выбрать **Save** в меню **File** или использовать соответствующий значок на панели инструментов). Вы всегда сможете вернуться к последнему сохраненному варианту.

Добавьте, где это необходимо двойные связи **L**-щелчком. Полуторные связи ароматического кольца обозначаются пунктирной линией. Их можно нарисовать при помощи двойного **L**-щелчка вблизи одной из внутренних сторон кольца.

Маркирование атомов

1. В меню **Display** нужно выбрать **Labels (Этикетки)**.



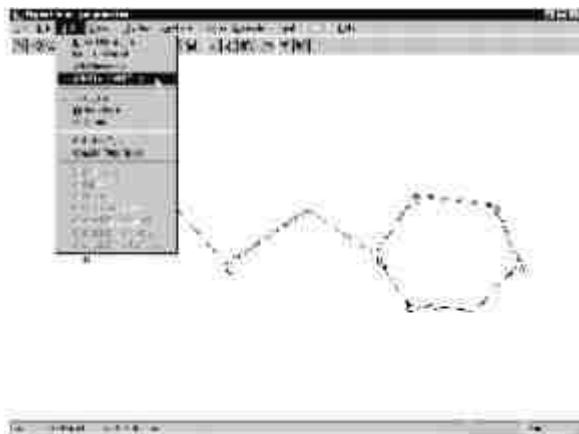
2. В появившемся блоке опции Атом автоматически отмечено **None**. Можно выбрать **Symbol, Name, Number** и пр., отметив нужное **L**-щелчком, и нажать на OK.
3. Блок закрывается и все атомы помечаются.



Редактирование индивидуальных атомов

Вся структура в рабочем окне построена из атомов углерода, но чтобы структура была верной, необходимо заменить один из углеродов на атом кислорода.

Для этого:

1. Откройте таблицу элементов и одним **L**-щелчком выберите кислород и закройте ящик. Кислород устанавливается как встроенный элемент.
2. Наведите курсор на атом углерода в конце алифатической цепи и щелкните по нему однократно левой кнопкой мыши. *Углерод С* (атом голубого цвета) заменится на *кислород О* (атом красного цвета). Теперь основная углеродная цепь выстроена.
3. Для того чтобы построенная схема приобрела правильную геометрию, в меню **Build** выберите **Model Build (Разработчик моделей)**. **Схема 2D** (двумерная) преобразуется в трехмерную структуру **3D**.



4. Если в процессе преобразования геометрии молекула была повернута или смещена на периферию экрана, то ее можно вернуть в первоначальное положение, используя соответствующие кнопки панели инструментов  (**Rotate out-of plane**,  **Translate**).
5. Для того чтобы автоматически добавить необходимое количество атомов водорода необходимо в меню **Build** выбрать пункт **Add Hydrogens (Добавить водороды)**
6. Если водороды не отображаются, то в меню **Display** отметьте **Show Hydrogens (Показать водороды)** и повторите команду **Add Hydrogens**.
7. Построенную таким образом структуру сохраните.

Расчеты в HyperChem

Setup (Меню установки)

Меню *Setup* задает метод расчета. Это может быть: молекулярная механика, полуэмпирические квантово-химические методы (10 вариантов, от расширенного метода Хюккеля до метода PM3) и неэмпирический метод Хартри-Фока (*ab initio*) в различных базисах.

Меню *Setup* содержит опции для проведения молекулярно-механических и квантово-химических расчетов энергетических, геометрических и электронных параметров молекулярных систем. После того, как Вы выбрали нужную опцию в меню *Setup*, используйте меню *Compute*, для проведения расчетов необходимых характеристик.

Программа ***HyperChem*** может выполнять расчеты энергии систем и их равновесной геометрии методом молекулярной механики с использованием четырех модельных потенциалов (**MM+**, **AMBER**, **BIO+** и **OPLS**), девятью полуэмпирическими квантово-химическими методами (Расширенный метод Хюккеля, **CNDO**, **INDO**, **MINDO3**, **MNDO**, **AM1**, **PM3**, **ZINDO/1** и **ZINDO/S**), или неэмпирическим (*ab initio*) методом квантовой химии в различных базисах.

Меню *Setup* имеет следующие пункты:

Molecular mechanics (Молекулярная механика)	Выберите опцию <i>Molecular mechanics</i> (Молекулярная Механика), чтобы использовать ньютоновский (а не квантово-химический) метод расчетов молекулярных потенциалов
Semi-empirical (Полуэмпирический)	Выбор полуэмпирического метода (<i>Semi-empirical</i>) позволит использовать один из квантово-химических методов расчета параметров молекулярных систем вместо молекулярной механики или <i>ab initio</i>
Ab Initio	Выбор <i>Ab Initio</i> (неэмпирический метод Хартри-Фока) позволит использовать этот метод квантовой химии вместо молекулярной механики или одного из полуэмпирических методов.
Periodic box (Периодический ящик)	Периодический ящик (<i>Periodic Box</i>) Выбор этого пункта позволит поместить молекулярную систему в периодический ящик, содержащий молекулы воды
Restraints (Ограничения)	Ограничения (<i>Restraints</i>) Выбор этого пункта позволяет добавлять граничные условия (действующие силы) для 1, 2, 3 и 4 выделенных типов атомов для молекулярной механики и квантовой химии. Этот пункт меню остается неактивным в случае, если такого выделения сделано не было.
Set velocity (Присвоение скорости)	Присвоение скорости (<i>Set Velocity</i>). Выбор этого пункта позволяет устанавливать скорости атомам, входящим в систему. Этот пункт используется в некоторых видах молекулярно-динамических расчетов.
Вычисления с использованием сети (Network)	Выбор этого пункта (<i>Network</i>) позволяет проводить расчеты с использованием удаленного сервера для молекулярно-механических, полуэмпирических и неэмпирических расчетов
Select parameter set (Выбор набора параметров)	Выбор этого пункта позволяет устанавливать альтернативный набор параметров для молекулярной механики
Compile parameter file (Компиляция файла параметров)	Выбор этого пункта позволяет преобразовать новый файл параметров из текстового вида или файла базы данных в двоичный код, используемый программой HyperChem
Reaction map (Карта реакции)	Выбор этого пункта позволяет строить карту реакции, т.е. путь от реагентов к продуктам реакции и синхронного поиска переходного состояния. Этот пункт остается неактивным до тех пор, пока не сделан выбор реагентов и продуктов реакции

Молекулярная механика (ММ)

Выбор в меню *Setup* пункта, соответствующего молекулярной механики, позволяет использовать классический Ньютоновский метод вычислений энергии одной точки, равновесной геометрии и молекулярной динамики объектов вместо квантово-механического подхода (одного из полуэмпирических методов или неэмпирического метода Хартри-Фока (*ab initio*)).

В методе молекулярной механики атомы рассматриваются как ньютоновские частицы, которые взаимодействуют друг с другом посредством неких потенциальных полей, задаваемых эмпирически. Потенциальная энергия взаимодействия зависит от длины связей, углов связи, торсионных углов и нековалентных взаимодействий (в т.ч. сил Ван-дер-Ваальса,

электростатических взаимодействий и водородных связей). В этих расчетах силы, действующие на атомы, представляются в виде функций координат атомов.

Примечание: Если в рабочей области выделена только часть системы, в расчет будут включаться взаимодействия только выделенной части. При оптимизации геометрии и расчетах методом молекулярной динамики, в этом случае, только атомы выделенной части будут менять свое положение в пространстве, тогда как невыделенные – нет, при этом в расчетах будет учитываться потенциальные взаимодействия между частями системы.



Для начала расчетов методом молекулярной механики в диалоговом окне необходимо выбрать *Force field* (Силовое поле) - потенциальную функцию для расчетов. Можно выбрать один из четырех методов (**MM+**, **AMBER**, **BIO+**, **OPLS**), ссылки на которые можно увидеть в диалоговом окне.

Метод **MM+** разрабатывался для органических молекул. Он учитывает потенциальные поля, формируемые всеми атомами рассчитываемой системы и позволяет гибко модифицировать параметры расчета в зависимости от конкретной задачи, что делает его, с одной стороны, наиболее общим, а с другой – резко увеличивает необходимые ресурсы по сравнению с другими методами молекулярной механики. Ряд возможностей для изменения параметров этого метода можно получить, выбрав кнопку *Options* в пункте выбора *Силового поля*.

Метод **AMBER** разрабатывался для белков и нуклеиновых кислот. В нем существует возможность выбрать опцию либо учета всех атомов по отдельности, либо опцию объединенного атома, под которым подразумевается группа эквивалентных атомов с одинаковыми свойствами. В последнем случае несколько атомов, либо их групп, обрабатываются как один атом с одним типом.

BIO+ разрабатывался для биологических макромолекул и во многом повторяет **AMBER**.

OPLS разработан для белков и нуклеиновых кислот. Он подобен **AMBER**, но более точно обрабатывает нековалентные взаимодействия.

Диалоговое окно молекулярной механики MM+ Options



Диалоговое окно MM+ содержит набор настроек для соответствующего силового поля.

Electrostatics (Электростатика) Нековалентные электростатические взаимодействия рассчитываются с использованием взаимодействий дипольного типа или частичных атомных зарядов.

- **Bond dipoles** используется для расчетов нековалентных электростатических взаимодействий. Значение этого параметра определяется в файле параметров MM+.

- **Atomic charges** используется для расчетов нековалентных электростатических взаимодействий. Вы можете задавать неполные (частичные) атомные заряды посредством меню *Build*, пункта *Set Charge* или Вы можете проводить полуэмпирические или *ab initio* расчеты, сначала рассчитывая частичные заряды для каждого атома методом Муликена.

Cutoffs (Отключение) этот параметр определяет минимальное расстояние для нековалентных взаимодействий.

- **Switched** вводит сглаживающую функцию при расчетах молекул в **Periodic Box (Периодический ящик)**. Этот подход позволяет плавно уменьшать слабые взаимодействия вплоть до нуля, перемещаясь из внутренней сферы во внешнюю. В этом случае **HyperChem** устанавливает параметр *Switched* и значения внутренней (*Inner*) и внешней (*Outer*) сфер (*Spheres*).

- **None**. Этот параметр устанавливается для расчета систем в вакууме.

- **Shifted** вводит сглаживающую функцию, которая действует на все пространство от 0 до внешней сферы. Эта функция позволяет плавно уменьшать нековалентные взаимодействия до 0.

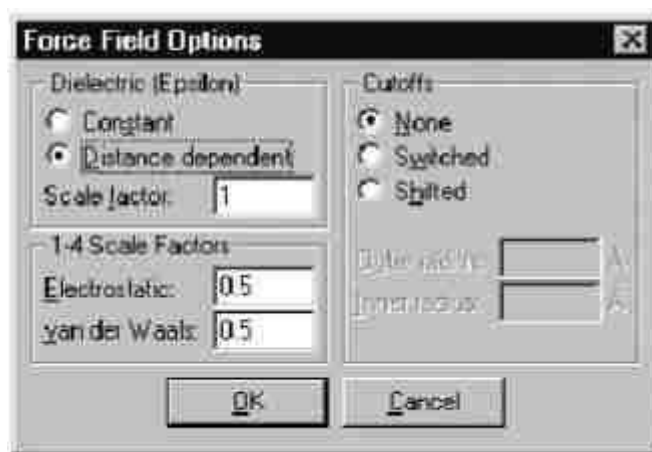
- **Outer radius** для параметров *Switched* и *Shifted* определяет минимальное расстояние, на котором нековалентные взаимодействия становятся равными 0. Обычно это значение выбирается не менее чем на 4 ангстрема больше, нежели чем внутренний радиус. Для периодических граничных условий это значение равно половине минимального размера периодического ящика.

- **Inner radius** выбирается только в случае установки *Switched cutoffs*. Это максимальное межатомное расстояние для полного учета нековалентных взаимодействий. В случае выбора периодических граничных условий это значение выбирается на 4 ангстрема меньше, нежели

чем половина минимального размера *Периодического ящика*, или менее, вплоть до 0. Внимание, установки *Cutoffs* возвращаются к своим стандартным значениям в случае, когда в рабочее поле помещается новая молекула.

Диалоговое окно опций силового поля (*Force Field Options Dialog Box*)

Это окно используется для выбора параметров силовых полей **AMBER**, **BIO+** и **OPLS**. *HyperChem* хранит значения этих параметров, исключая параметры *Cutoffs*, в Registry или в файле chem.ini и использует их для последующих вычислений.



Dielectric permittivity (epsilon) (диэлектрическая постоянная). Параметры **Constant** (Постоянная) или **Distance dependent** (Зависящая от расстояния) определяют методы расчета диэлектрической постоянной ϵ , фактора, который модифицирует взаимодействие зарядов (и электростатического потенциала).

- **Constant** (Постоянная). Выбор этого параметра делает диэлектрическую постоянную константой и соответствует периодическим граничным условиям *Периодического ящика*. Выбор этого пункта соответствует веществу, находящемуся в газовой фазе, либо в идеальном растворе.

- **Distance dependent** (Зависящая от расстояния). Выбор этого параметра делает ϵ пропорциональной межатомному расстоянию. Подобный подход аппроксимирует эффект сольватации в отсутствии идеального растворителя и позволяет ускорять расчеты. Данный параметр рекомендуется использовать при расчетах методом **OPLS**. Так как данный параметр моделирует присутствие сольвента, его не следует применять, когда молекулы сольвента присутствуют в моделируемой системе.

В случае выбора параметра **Constant** ϵ (ϵ)=(диэлектрическая постоянная свободного пространства) * (масштабный множитель (*Scale factor*)). В случае выбора параметра **Distance dependent** ϵ (ϵ)=(диэлектрическая постоянная свободного пространства) * (масштабный множитель (*Scale factor*)) * (межатомное расстояние). Масштабный множитель должен быть ≥ 1 . По умолчанию он принимается равным 1, что удовлетворяет для большинства рассчитываемых систем.

1-4 Scale factor (Масштабный множитель 1-4) нековалентные взаимодействия между атомами, разделенными в точности тремя связями, умножаются на этот множитель.

- **Electrostatic (Электростатика)** модифицирует силу взаимодействия зарядов между атомами, разделенными тремя связями. Этот параметр меняется в пределах от 0 до 1. Для силового поля **AMBER** и **OPLS** необходимо использовать 0.5, для **BIO+** рекомендуется 1.0, 0.5 или 0.4 в зависимости от набора других параметров.

- **Van-der-Waals (Ван-дер-Ваальс)** модифицирует ван-дер-ваальсовы взаимодействия между атомами, разделенными тремя связями, меняется в пределах от 0 до 1. Для силового поля **AMBER** необходимо использовать 0.5, для **OPLS** – 0.125, для **BIO+** - 1.0.

Cutoffs (Отсечения) определяет расстояние, после которого нековалентные взаимодействия между атомами не учитываются. Его необходимо вводить для того, чтобы избежать учета взаимодействия с соседями по периоду в случае расчетов в *Periodic Box*.

Полуэмпирические методы расчета электронной структуры (Semi-empirical)

Электронную структуру исследуемых молекул в программе **HyperChem** можно рассчитывать способами: используя полуэмпирические методы расчета, либо – неэмпирический метод Хартри-Фока, сделав выбор в меню *Setup*.



Полуэмпирические методы расчета можно использовать для всех типов расчетов в меню *Compute*. Полуэмпирические методы решают уравнение Шредингера для атомов и молекул с использованием определенных приближений и упрощений. Все методы этой группы характеризуются тем, что: расчет ведется только для валентных электронов; пренебрегаются интегралы определенных взаимодействий; используются стандартные не оптимизированные базисные функции электронных орбиталей и используются некоторые параметры, полученных в эксперименте. Экспериментальные параметры устраняют необходимость расчетов ряда величин и корректируют ошибочные результаты приближений. Необходимо помнить, что полуэмпирические методы в программе **HyperChem** могут обрабатывать не все элементы таблицы Менделеева, а только те, параметры которых внесены в файлы параметров.

Большинство доступных в программе **HyperChem** полуэмпирических методов включают схему для устранения вычислений, которые происходят со значительными затратами

процессорного времени, в основном – расчета ряда интегралов перекрывания, а метод **INDO** (**Intermediate Neglect of Differential Overlap**) (см. далее) не вычисляет и интегралы расталкивания, которые должны иметь небольшие величины.

HyperChem также позволяет Вам рассчитывать электронную структуру только части системы, используя смешанные методы вычисления. Например, можно изучить электронную структуру активного центра белка с использованием полуэмпирических методов расчета, учитывая оставшуюся часть белка и молекул растворителя в рамках метода молекулярной механики. Для этого, перед тем, как начинать расчет, выделите нужную часть системы с использованием инструментария меню *Select*, а затем, введите соответствующие параметры меню *Setup* и *Compute*. Необходимо подчеркнуть, что такие расчеты возможно проводить только в том случае, если выделенная часть системы не соединена формальными химическими связями с остальной частью молекулярной системы. (Например, построив модель белка можно удалить соответствующие химические связи активного центра, электронную структуру которого мы должны исследовать, а затем выделить активный центр с использованием различных способов меню *Select*. Например, выбрать параметр *Molecules* и выделить активный центр одним *L-нажатием*, либо выделить в нужной части один атом, а затем выбрать пункт *Extend to sp3* в меню *Select*, при этом будет выделена вся молекулярная система, в которую входит выбранный атом. В этом случае программа **HyperChem** квантово-химически рассчитывает только выделенную часть атомов, а остальные рассматривает только как некий потенциал. В процессе оптимизации геометрии координаты не выделенной части атомов являются фиксированными и не изменяются в ходе проведения расчетов.

Расширенный метод Хюккеля (Extended Huckel) (PMX) предназначен для вычислений молекулярных орбиталей и не позволяет оптимизировать геометрию и проводить молекулярно-динамические расчеты. В нем используется приближение невзаимодействующих электронов и в нем не используется приближение самосогласованного поля (**SCF**)

Метод CNDO (Complete Neglect of Differential Overlap, полное пренебрежение дифференциальным перекрыванием) является простейшим методом **SCF**. Он используется для расчетов основного состояния электронных характеристик систем с открытой и закрытой оболочками, оптимизации геометрии и полной энергии.

Метод INDO (Intermediate Neglect of Differential Overlap, частичное пренебрежение дифференциальным перекрыванием) улучшает метод **CNDO** за счет учета расталкивания электронов на одном атомном центре. Позволяет проводить расчет основного состояния систем с открытой и закрытой оболочками, оптимизации геометрии и полной энергии. Это **SCF** метод.

Метод MINDO3 (Modified INDO, version 3, улучшенный метод INDO, версия 3) является дальнейшим развитием и расширением метода **INDO**. Для многих взаимодействий в нем используются эмпирические параметры вместо соответствующих вычислений. Этот метод позволяет получать хорошие результаты для больших органических молекул при расчетах основного состояния систем с открытой и закрытой оболочками, оптимизации геометрии и полной энергии. Это **SCF** метод.

Метод MNDO является дальнейшим развитием метода **MINDO3**, в котором исправлен ряд ошибок последнего. Позволяет проводить качественные расчеты электронной и атомной структур органических молекул, содержащих атомы 1-й и 2-й главных подгрупп (но не атомов переходных элементов). Этот метод позволяет получать хорошие результаты для

больших органических молекул при расчетах электронных характеристик системы и теплот образования. Это **SCF** метод.

Метод AM1 является улучшением метода **MNDO**. Один из наиболее точных методов. Используется для органических молекул, содержащих элементы из главных подгрупп 1 и 2 групп периодической системы. Возможно, этот метод позволяет получать более качественные результаты, по сравнению с методом **MNDO**, для молекул, содержащих как азот, так и кислород. Вычисляет электронную структуру, оптимизирует геометрию, рассчитывает полную энергию и теплоты образования. Это метод **SCF**.

Метод PM3 является версией метода **AM1**. **PM3** отличается от **AM1** только величинами параметров. Параметры для **PM3** были получены сравнением большого числа и вида экспериментов с результатами расчетов. Как правило, нековалентные взаимодействия в методе **PM3** являются менее расталкивающими, нежели чем в **AM1**. **PM3** первоначально предназначался для расчета для органических молекул, но потом он был также параметризован и для ряда других групп элементов, в частности – и для переходных металлов. Это метод **SCF**.

Метод ZINDO/1 является вариантом метода **INDO**, адаптированного для проведения расчетов молекул, включающих атомы переходных элементов. Эквивалентен последней версии метода **INDO/1**, который отличается от оригинала использованием постоянных орбитальных экспонент. **ZINDO/1** позволяет вычислять энергетику и геометрию молекул, содержащих переходные металлы.

Метод ZINDO/S является версией метода **INDO**, параметризованного для воспроизведения УФ и видимых оптических переходов при расчетах конфигурационного взаимодействия (*CI*) с одночастичными возбуждениями. Полезен для прогнозирования УФ и видимых спектров, но не пригоден для оптимизации геометрии или молекулярной динамики.

CNDO, INDO

[illegible]

MINDO3

[illegible]

MINDO

H																					He
Li	Be										B	C	N	O	F	Ne					
Na	Mg										Al	Si	P	S	Cl	Ar					
K	Ca									Zn	Ga	Ge	As	Se	Br	Kr					
Rb	Sr									Cd	In	Sn	Sb	Te	I	Xe					

AMEI

[illegible]

PM3

[illegible]

ZINDO/1

H																	He
Li	Be											B	C	N	O	F	Ne
Na	Mg											Al	Si	P	S	Cl	Ar
K	Ca	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn	Ga	Ge	As	Se	Br	Kr
Rb	Sr	Y	Zr	Nb	Mo	Tc	Ru	Rh	Pd	Ag	Cd	In	Sn	Sb	Te	I	Xe

ZINDO/S

H																	He
Li	Be									B	C	N	O	F	Ne		
Na	Mg									Al	Si	P	S	Cl	Ar		
K	Ca	Sc	Ti	V	Cr	Mn	Fe	Co	Ni	Cu	Zn	Ga	Ge	As	Se	Br	Kr
Rb	Sr	Y	Zr	Nb	Mo	Tc	Ru	Rh	Pd	Ag	Cd	In	Sn	Sb	Te	I	Xe

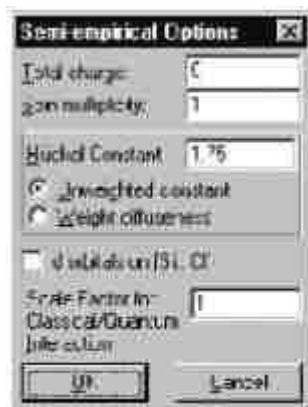
Для установки параметров полуэмпирических расчетов необходимо открыть диалоговое окно выбранного полуэмпирического метода. Есть две версии этого диалогового меню (смотри последующие разделы).

В этих таблицах представлены химические элементы, которые могут рассчитываться теми или иными полуэмпирическими методами. В том случае, если символ элемента присутствует в таблице, это означает, что в программе *HyperChem* для данного полуэмпирического метода существует часть для расчета интегралов с участием валентных электронов этого элемента с соответствующими главным и орбитальным квантовыми числами. Однако, в настоящее время не для всех атомов, для которых существует программная реализация того или иного полуэмпирического метода, известен набор параметров. Если таковые становятся известными, то можно изменять соответствующие файлы параметров. Элементы, для которых параметры известны и внесены в соответствующие файлы параметров, в этих таблицах закрашены.

Диалоговое окно полуэмпирического метода

Диалоговое меню метода PMX

В расширенном методе Хюккеля используется отличное от всех остальных полуэмпирических методов диалоговое окно.



Total charge (Полный заряд системы) вычисляется как разность между полным количеством электронов в системе и суммарным зарядом ядер. Целочисленный и целочисленный положительный для катионов и отрицательный - для анионов.

Spin multiplicity (Мультиплетность по спину) вычисляется как $2S+1$, где S – полный спин системы. Каждый неспаренный электрон имеет спин, равный $1/2$. Системы с закрытой оболочкой (синглет) имеют мультиплетность, равную 1. Обладающие одним неспаренным электроном (дублет) и двумя (триплет) – 2 и 3 соответственно. В это диалоговое окно можно вводить величины от 1 до 6.

Huckel constant (Константа Хюккеля) пропорциональности между диагональными и недиагональными матричными элементами. Стандартное значение равно 1.75. Более высокие значения увеличивают вес перекрывания атомных орбиталей в определении полной энергии, а меньшие – одноэлектронных энергий.

- **Unweighted constant (Не взвешенная константа)** выбор этого пункта означает, что хюккелевская константа используется в расчетах без изменений.

- **Weight diffuseness (Вес диффузности)** умножает хюккелевскую константу на множитель, который учитывает диффузность атомных орбиталей, что встречается достаточно редко для органических молекул и молекул, состоящих из атомов главных подгрупп.

- **d-orbitals on ... (d-орбитали)** этот пункт позволяет учитывать d -орбитали для атомов Si, P, S, Cl.

Scale factor (Масштабный множитель) масштабирует введение классических частичных зарядов в случае проведения смешанных (молекулярно-механических и квантово-химических) расчетов (Для более детального ознакомления см. *HyperChem Computation Chemistry, Theory and Methods*).

Диалоговое окно других полуэмпирических методов



При выборе одного из полуэмпирических методов, перечисленных выше (исключая **PMX**), возникает соответствующее диалоговое окно.

Charge and Spin

- **Total charge (Полный заряд системы)** вычисляется как разность между полным количеством электронов в системе и суммарным зарядом ядер. Целочисленный и целочисленный положительный для катионов и отрицательный - для анионов.
- **Spin multiplicity (Мультиплетность по спину)** вычисляется как $2S+1$, где S – полный спин системы. Каждый неспаренный электрон имеет спин, равный $\frac{1}{2}$. Системы с закрытой оболочкой (синглет) имеют мультиплетность, равную 1. Обладающие одним неспаренным электроном (дублет) и двумя (триплет) – 2 и 3 соответственно. В это диалоговое окно можно вводить величины от 1 до 6.

State (Состояние). Этот параметр описывает возбужденные состояния валентных электронов в системе.

- **Lowest (Наинизшее)** – выбор этого параметра означает, что программа будет выбирать низшее из всех возможных электронных состояний в системе с заданной мультиплетностью по спину.
- **Next Lowest (Первое возбужденное)** – выбор этого параметра означает, что программа будет рассчитывать первое возбужденное электронное состояние с заданной мультиплетностью по спину.
- **Convergence Limit (Параметр сходимости).** SCF расчет заканчивается тогда, когда отличия в полной энергии двух последующих итераций становятся меньше некоего заранее заданного значения. По умолчанию этому значению присваивается значение 0.01 ккал/моль. Этот параметр может меняться от 1 до 0.001. Параметр сходимости 1 ккал/моль является очень грубым, а 0.001 ккал/моль – не всегда достижим, так как систематическая ошибка полуэмпирических методов достигает примерно такой же величины. При поиске переходного состояния рекомендуется задавать минимальный параметр сходимости.
- **Iteration limit (Предельное количество итераций).** Этот параметр определяет предельное количество итераций на шаге самосогласования. Рекомендуемое количество – 50, но можно, в случае медленной сходимости, ставить и большее число – порядка 100 или 200, например – в случае поиска переходного состояния.
- **Accelerate convergence (Ускорение сходимости).** Выбор этого параметра убыстряет сходимость SCF расчетов. При этом *HyperChem* включает процедуру, известную как “Прямое инвертирование подпространства итераций” (*Direct Inversion of Iterative Subspace, DIIS*) (Более подробно смотри *HyperChem Computational Chemistry*).

Spin pairing (Спиновое состояние). Возможно выбрать два метода расчета спиновых состояний молекул. Первый – неограниченный метод Хартри-Фока (*Unrestricted Hartree-Fock method, UHF*) и ограниченный метод Хартри-Фока (*Restricted Hartree-Fock method, RHF*).

- **UHF** рассматривает спин-орбитали с различным пространственным распределением для a и b орбиталей. Этот метод применяется при изучении систем, как с открытыми, так и с закрытыми электронными оболочками. Так, для последних он хорошо описывает реакции диссоциации. Однако, из-за удвоения количества орбиталей, время расчета этим методом увеличивается вдвое. У этого метода существуют и другие ограничения, связанные с его основами.

- В **RHF** считается, что электроны с различным спином занимают одинаковые, в смысле пространственного распределения, орбитали. При этом неспаренные электроны тоже могут занимать отдельные орбитали. Этот метод применяется как для открытых, так и для закрытых электронных оболочек.

Overlap Weighting Factors (Коэффициент масштабирования перекрывания).

Дополнительные параметры для двух **ZINDO** методов, которые способны изменять вклады σ и π связей. Более подробно этот параметр описан в *HyperChem Computational Chemistry, Theory and Methods*.

- **Sigma-Sigma** определяет s - s перекрывание атомных орбиталей. Обычно он равен 1.0 для **ZINDO/1** и 1.67 для **ZINDO/S**.

- **Pi-Pi** определяет вес s - s перекрывания атомных орбиталей. Он равен 1.0 для **ZINDO/1**. Для **ZINDO/S** этот параметр равен 0.640 при расчетах комплексов переходных металлов и 0.585 при расчетах органических молекул.

Configuration Interaction (Конфигурационное взаимодействие). Эта опция используется для активации расчета конфигурационных взаимодействий и открывает соответствующее диалоговое окно. Такой подход необходимо применять при расчетах УФ и оптических спектров в видимом диапазоне. Выбор этой опции существенно увеличивает время расчетов.

Диалоговое окно Configuration Interaction

Учет конфигурационного взаимодействия может быть использован для улучшения качества волновой функции и энергии состояния. Все расчеты в приближении самосогласованного поля (**SCF**) основаны на одноэлектронной модели, суть которой заключается в том, что каждый электрон движется в усредненном поле, которое формируется всеми остальными электронами. Считается, что электроны взаимодействуют мгновенно и стремятся избегать друг друга согласно принципу Паули. Такая корреляция приводит к понижению среднего межэлектронного отталкивания и, в свою очередь, к понижению энергии состояния. Отличие между полной энергией, рассчитанной в **SCF** подходе и энергией, полученной в точно нерелятивистском подходе, называется корреляционной энергией.

Существуют два типа электронных корреляций: статические и динамические. Статические корреляции связаны с энергетическим вырождением данного состояния, а динамические – со стремлением электронов избегать друг друга, что происходит с бесконечно большой скоростью.

КВ (CI) расчеты, возможно, являются наиболее широко распространенным методом выхода за пределы **SCF**-подхода. Результатом **SCF** расчета является конфигурация состояния, в котором одноэлектронные уровни жестко заполнены электронами. Другие конфигурации могут быть сформированы из конфигурации, полученной в самосогласованном расчете при помощи возбуждения электронов с занятых на виртуальные (вакантные) орбитали.

Результатом **КВ** расчета является набор улучшенных состояний, каждое из которых представляется линейной комбинацией таких конфигураций. **КВ** расчеты невозможно проводить в режиме оптимизации геометрии. В методе **PMX** этот подход также не реализован.

Для установки параметров **КВ** расчетов используется диалоговое окно *Configuration Interaction* (Конфигурационного взаимодействия). Для этого необходимо выбрать соответствующую кнопку в диалоговом меню полуэмпирических методов.



Затем нужно выбрать один из параметров: **None** (Ни одного), **Singly Excited** (Однократно возбужденное) или **Microstate** (Микросостояние).

- **None** - расчет конфигурационных взаимодействий производиться не будет.
- **Singly Excited** - в расчете будут учитываться только однократно возбужденные состояния.
- **Microstate** означает, что в расчете кроме однократно-возбужденных состояний будут учитываться и все возможные многократные.

Orbital Criterion (Орбитальный критерий). Выбор этого параметра определяет диапазон орбиталей, с которых и на которые происходят электронные возбуждения, формирующие взаимодействующие конфигурации.

- **Occupied** (Занятые) определяет область занятых орбиталей, начиная с высшей занятой молекулярной орбитали (**HOMO**), с которой происходит возбуждение орбиталей.
- **Unoccupied** (Вакантные) определяет область вакантных (виртуальных) орбиталей, начиная с низшей вакантной орбитали (**LUMO**), на которые происходят электронные возбуждения.

Energy Criterion (Энергетический критерий) является опцией Орбитального критерия, который устанавливает ограничения по энергии при генерировании набора взаимодействующих конфигураций. Эта опция доступна только для Однократно возбужденных конфигураций.

- **Maximum Excitation** (Максимальное возбуждение) определяет наибольшую разницу по энергии в эВ между занятыми и вакантными орбиталями, включенными в **CI** расчет. В общем виде, конфигурации с высокой энергией не могут сильно взаимодействовать с конфигурацией основного состояния. Чем выше этот параметр, тем больше конфигураций включается в **CI** расчет.

Практические применения метода конфигурационного взаимодействия.

CI расчеты возможно использовать при расчетах:

- УФ и видимых спектров,
- Энергии возбужденных состояний,

- Изучения создания или разрыва химических связей (например, диссоциация H_2), изменения спинового состояния,
- Изучения эффектов, связанных с дисперсионными силами Лондона,
- Описания вырожденных, или близких к вырождению состояний,
- Изучения расщепления синглет-триплет на более высоком уровне.

Метод микросостояний понижает энергию некоррелированного состояния так же, как и возбужденных состояний. Метод однократно возбужденного **CI** предназначен только для расчетов УФ и видимых спектров и не улучшает энергию. Основного состояния (теорема Бриллюэна).

При использовании орбитального критерия для симметричных систем, для того, чтобы получить корректные результаты, необходимо включить либо все, либо ни одного из наборов вырожденных орбиталей. Необходимо также внимательно использовать Энергетический критерий. Этот критерий должен быть больше, нежели чем энергетическая щель между занятыми и вакантными орбиталями.

В больших системах, как правило, в небольших энергетических интервалах находится большое количество орбиталей. Следовательно, размер **CI** матрицы может быть очень чувствительным к величине энергетического критерия. Так как время вычислений сильно зависит от размера **CI** матрицы, необходимый вычислительный ресурс, особенно если использовать методы **MNDO**, **AM1** или **PM3**, может стать не приемлемо большим. Для того чтобы избежать такой ситуации, необходимо тщательно проанализировать результаты **RHF** расчета.

Расчеты в смешанном режиме (Квантовая механика/квантовая химия)

При расчетах в смешанном режиме, программа *HyperChem* выбранную часть системы часть рассчитывает квантово-механически, а остальную – молекулярно-механически. Программа позволяет проводить подобные расчеты с использованием всех полуэмпирических квантово-химических методов. Необходимо напомнить, что при оптимизации геометрии системы только выделенная часть атомов будет менять свои координаты в ходе оптимизации. Остальные атомы будут вносить свой вклад лишь как некое статичное поле, генерируемое зарядами на них, вычисленными или присвоенными им ранее.

Для того чтобы производить расчеты в этом режиме, необходимо выделить ту часть атомов молекулы, которая будет рассчитываться квантово-механически. В случае, если некоторые из молекул лишь частично выделены, необходимо убедиться, что граничные атомы связаны с остальной частью *sp³*-связями. Для того, чтобы убедиться в этом, можно использовать параметр *Extend to sp³* меню *Select*, до того, как запускать полуэмпирический расчет. Эта опция распространит выделение рассчитываемой области по всем направлениям до тех пор, пока выделенная часть не достигнет конца молекулы, либо не найдет *sp³-sp³* связь.

Неэмпирический (ab initio) метод Хартри-Фока

Выбор параметра *ab initio* в меню *Setup* позволяет проводить неэмпирические расчеты электронной и атомной структур объектов. В отличие от молекулярно-механических и полуэмпирических методов, неэмпирический *метод Хартри-Фока* не требует для проведения расчетов знания каких-либо эмпирических параметров, например – силы и

длинны отдельных связей, значений интегралов перекрывания и пр. В меню *Setup* этот пункт стоит третьим.

Ab initio метод требует для своих расчетов гораздо больше вычислительных ресурсов, нежели чем молекулярно-механические и полуэмпирические методы. Особенно это касается оптимизации геометрии или проведения молекулярно-динамических расчетов. Для оптимизации геометрии рекомендуется, на начальном этапе использовать молекулярную механику, затем – один из полуэмпирических методов, для того, чтобы получить более или менее обоснованную начальную геометрию. Однако для ряда неорганических систем молекулярно-механические и полуэмпирические расчеты дают некорректные результаты, поэтому рекомендуется использовать параметр *Model Bilder*, для того, чтобы получить более или менее подходящую стартовую геометрию.

Выбор базисного набора

Любой набор одноэлектронных волновых функций может служить базисным набором (или просто – базисом) для ЛКАО (линейная комбинация атомных орбиталей, английская аббревиатура – **LCAO**) приближения. Однако, хорошо определенный базис будет предсказывать электронные свойства системы с использованием гораздо большего числа членов, нежели, чем плохо определенный. Следовательно, выбор наиболее подходящего базисного набора в **ab initio** расчете является критичным для точности и обоснованности результатов. В программе **HyperChem** определен формат файла базисных наборов (расширение *.BAS), в который включены целый ряд стандартных базисных наборов. Тем не менее, пользователь может сам определить необходимые для расчетов базисные наборы. Много обычных и широко используемых базисных наборов автоматически поддерживаются в **HyperChem**. Эти наборы включают в себя:

- STO-1G and STO-1G* (H до He) [1] ;
- STO-2G and STO-2G* (H до Xe) [1] ;
- STO-3G and STO-3G* (H до Xe) [1] ;
- STO-4G and STO-4G* (H до Xe) [1] ;
- STO-5G and STO-5G* (H до Xe) [1] ;
- STO-6G and STO-6G* (H до Xe) [1] ;
- 3-21G, 3-21G*, and 3-21G** (H до Ar) [2] ;
- 4-21G, 4-21G*, and 4-21G** (H до Ne) [3] ;
- 6-21G, 6-21G*, and 6-21G** (H до Ar) [2] ;
- 4-31G, 4-31G*, and 4-31G** (H до Ne) [3] ;
- 5-31G, 5-31G*, and 5-31G** (H до F) [3] ;
- 6-31G, 6-31G*, and 6-31G** (H до Ar) [3] ;
- 6-311G, 6-311G*, and 6-311G** (H до Ar) [4] ;
- D95, D95* and D95** (H до Cl) [5] .

Диалоговое окно метода *ab initio*



В программе **HyperChem** возможно использовать много базисных наборов. В этом диалоговом окне кнопка **Apply Basis Set** служит для того, чтобы установить выбранный базис или для всего объекта, или для выделенной части, если такое выделение было сделано. Например, некоторые тяжелые атомы должны описываться базисом **6-31G** (без *d*-функций), тогда как другие – базисом **6-31G*** (с учетом *d*-функций). Параметр **Basis Set** диалогового меню приписывает соответствующий базис или всей молекуле, либо выделенной части.

- Выбор параметра **NoBasis Set** означает, что данному атому не будет приписываться ни одной базисной функции. Эта опция может быть использована только в том случае, если необходимо описать систему или выделенную часть с использованием дополнительных базисных функций.

- **Minimal (STO-3G)** приписывает минимальный **STO-3G** базис. Другие кнопки этого меню выбирают те базисные наборы, которые там указаны.

- **Other** позволяет активизировать кнопку **Assign Other Basis Set**, для того, чтобы использовать другие (не обозначенные в этом меню) базисные наборы.

Assign Other Basis Set. Нажатие этой кнопки приводит к вызову соответствующего меню, которое содержит полный список базисных наборов (исключая те, которые были приведены в предыдущем меню).

Extra Basis Function (Дополнительные базисные функции). Нажатие этой кнопки приводит к появлению соответствующего меню, которое позволяет вводить дополнительные базисные функции для выбранных атомов.

Options (Параметры расчета). Нажатие этой кнопки приводит к вызову соответствующего диалогового окна, при помощи которого задаются основные параметры *ab initio* расчета.

Advanced options служит для вызова соответствующего меню, в котором содержатся параметры, влияющие на процесс расчетов.

Кнопка **Apply Basis Set** присваивает всей молекуле или выделенной части атомов выбранный ранее базис. Кнопка **OK** служит для сохранения выбранных параметров и закрытия диалогового окна. Кнопка **Cancel** приводит к закрытию диалогового окна без сохранения выбранных параметров. Как уже говорилось ранее, в программе **HyperChem** существует возможность определять различные базисные наборы для разных частей рассчитываемой системы.

Диалоговое окно выбора других базисных наборов (Assign Other Basis Set Dialog Box)



Использование этого диалогового окна позволяет вызывать полный список базисных наборов, исключая те, которые уже были обозначены в диалоговом окне *Ab Initio Method*.

Выбор любого из этих базисов и последующее нажатие на кнопку ОК приведет к тому, что в соответствующем окне *Ab Initio Method* появится текстовая идентификация сделанного выбора. Все стандартные базисные наборы, как это уже говорилось ранее, хранятся в соответствующих файлах с расширением **.BAS**. Для того чтобы создать соответствующую ссылку на свой базисный набор в этом диалоговом окне, необходимо создать для него точку входа в секции **[basisset] Registry** или в соответствующем файле **CHEM.INI**.

Диалоговое окно параметров ab initio (Options)

Это диалоговое окно используется для выбора основных параметров неэмпирических вычислений. Эти параметры аналогичны параметрам полуэмпирических методов, описание которых было дано выше.



- **Gradient** задает расчет градиентов (первых производных полной энергии по атомным координатам). **RMS градиент** дает представление об отклонениях от оптимальной геометрии рассчитываемого объекта. Эта опция доступна только в режиме расчета одной точки. Это связано с тем, что для расчета этих параметров необходимо рассчитывать много двухэлектронных интегралов и их производных, что требует много процессорного времени, а в этом режиме необходимость таких расчетов может отсутствовать. Процедура **HyperGauss**, которая осуществляет неэмпирические расчеты, всегда рассчитывает градиенты при оптимизации геометрии, при молекулярно-динамических расчетах и расчетах молекулярных колебаний.

- **MP2 Correlation Energy** задает расчет корреляционной энергии в рамках теории возмущения Меллера-Плессета второго порядка. Эта опция тоже активна только для расчетов одной точки. Такие расчеты тоже увеличивают процессорное время, необходимую оперативную память и используемое дисковое пространство, так как для этого требуется переход от двухэлектронных интегралов на атомных орбиталях к таковым, рассчитанным уже на молекулярных орбиталях.

Это диалоговое окно имеет также параметры **Charge** и **Spin multiplicity**. Они служат для того, чтобы задавать полный электронный заряд системы, который определяется как разность между количеством электронов и суммарным ядерным зарядом и мультиплетность системы, которая определяется как $2S+1$ и может быть синглетом (1), дублетом (2), триплетом (3) и кваттетом (4).

SCF Controls (Секция параметров SCF расчета)

Эта часть диалогового меню служит для задания требуемой точности расчета **SCF** волновой функции и максимального количества итераций для достижения этой точности.

- **Convergence (Limit Предел сходимости)** служит для того, чтобы остановить **SCF** процедуру, когда разница в энергиях между двумя последующими итерациями становится меньше заданной величины. Для *ab initio* расчетов это значение, как правило, выбирается равным 0.00001 ккал/моль, а используемый параметр может лежать в интервале от 1 до 0.00000001 ккал/моль. Тем не менее, если этот параметр задать меньше 10^{-10} , то сходимости можно не достигнуть, так как сам метод Хартри-Фока дает соразмерную ошибку. При поиске переходных состояний рекомендуется использовать более жесткие критерии сходимости.

- **Iteration Limit (Предельное количество итераций)** определяет максимальное количество итераций в **SCF** расчете. Расчет останавливается, в случае если программа выполнила заданное количество итераций и при этом не достигла сходимости. В этом случае результат расчета может быть не верен, так как энергия может быть далека от истинной, либо она осциллировала в ходе расчета. 50 итераций представляется разумным для большинства случаев, тогда как задание большего количества (скажем, до 200) может быть оправданным при расчетах переходных состояний. В случае, когда расчет не может сойтись и при большем количестве итераций, то это, как правило, означает, что дальнейшее увеличение количества итераций не может изменить, так как система не сходится. Лишь в некоторых случаях - **Accelerate Convergence (Алгоритм ускорения сходимости)** может исправить ситуацию. Выбор этого параметра убыстряет сходимость **SCF** расчетов. При этом *HyperChem* включает процедуру, известную как “Прямое инвертирование подпространства итераций” (*Direct Inversion of Iterative Subspace, DIIS*) (Более подробно смотри *HyperChem Computational Chemistry*), которая может потребовать большего объема памяти. Включение этой опции может увеличить время, затрачиваемое на одну итерацию из-за того, что в этом подходе матрица Фока вычисляется как линейная комбинация текущей матрицы Фока и матриц Фока от предыдущих итераций. Как правило, такой подход уменьшает общее количество требуемых итераций.

Spin pairing (Спиновое состояние). Возможно выбрать два метода расчета спиновых состояний молекул. Первый – неограниченный метод Хартри-Фока (**Unrestricted Hartree-Fock method, UHF**) и ограниченный метод Хартри-Фока (**Restricted Hartree-Fock method, RHF**).

- **UHF** рассматривает спин-орбитали с различным пространственным распределением для *a* - и *b*- орбиталей. Этот метод применяется при изучении систем, как с открытыми, так и с закрытыми электронными оболочками. Так, для последних он хорошо описывает реакции диссоциации. Однако, из-за удвоения количества орбиталей, время расчета этим методом

увеличивается вдвое. У этого метода существуют и другие ограничения, связанные с его основами.

- В **RHF** считается, что электроны с различным спином занимают одинаковые, в смысле пространственного распределения, орбитали. При этом неспаренные электроны тоже могут занимать отдельные орбитали. Этот метод применяется как для открытых, так и для закрытых электронных оболочек.

Configuration Interaction (Конфигурационное взаимодействие). Эта опция используется для активации расчета конфигурационных взаимодействий и открывает соответствующее диалоговое окно. Такой подход необходимо применять при расчетах УФ и оптических спектров в видимом диапазоне. Выбор этой опции существенно увеличивает время расчетов.

Кнопка OK служит для сохранения выбранных параметров и закрытия диалогового окна. Кнопка Cancel приводит к закрытию диалогового окна без сохранения выбранных параметров. Как уже говорилось ранее, в программе **HyperChem** существует возможность определять различные базисные наборы для разных частей рассчитываемой системы.

Необходимо отметить, что **HyperChem** не поддерживает *ограниченный метод Хартри-Фока для систем с открытыми оболочками (ROHF)* на **ab initio** уровне.

Диалоговое окно параметров высокого уровня метода **ab initio** (**Ab Initio Advanced options**)

Это диалоговое окно служит для более точной настройки параметров **ab initio** расчетов.



Integral Format

- **Regular** определяет использование обычного формата для записи двухэлектронных интегралов. **HyperChem** использует 16 байт для записи каждого из интегралов. Первые 8 байт хранят 4 индекса интеграла, а последние 4 – его значение. Программа сохраняет эти величины только в том случае, если абсолютное значение интеграла больше или равно параметру **Cutoff**. В противоположном случае значение этого интеграла приравнивается к 0. Двухэлектронные интегралы и их индексы сохраняются на диске без модификации при выборе опции **Regular** и могут быть записаны в **.log-файл** при соответствующем выборе параметра **QuantumPrintLevel** меню **StartLog**.

- **Raffenetti** определяет использование формата Раффенети (*R.C. Raffenetti, Chem.Phys.Lett., 20, 335 (1973)*), который позволяет более просто формировать матрицу Фока в ходе **SCF** расчета. Этот формат, как правило, требует больше памяти и больше дискового пространства, однако позволяет повышать скорость расчета. Этот формат нельзя использовать при проведении **MP2**-расчетов.

Параметр **Cutoff** позволяет сохранять на диске только те интегралы, абсолютное значение которых равно или превышает задаваемый параметр. По умолчанию он равен 10^{*-10} Хартри. Этот параметр контролирует осуществление **SCF**-итераций, точность волновых функций и энергии, так как он уменьшает количество рассчитываемых двухэлектронных интегралов.

Параметр **Buffer size (Размер буфера)** определяет размер операционной памяти (в словах двойной точности, 8 байт для одного слова), которая требуется для хранения двухэлектронных интегралов до того, как записать их во временный на жестком диске (выбор этого диска может быть сделан в меню **File/Preferences/Path**). Более большой размер буфера способен уменьшить расчетное время из-за того, что программа будет обращаться к диску более редко. Если этот буфер будет достаточно велик, то **HyperChem** не будет обращаться к диску вовсе. Необходимо отметить, что в случае, когда программа останавливается из-за сбоя компьютера, либо по другой некорректной причине, этот временный файл остается на диске. Для того чтобы освободить это место, его необходимо удалять вручную.

Direct SCF calculation вычисляет двухэлектронные интегралы на каждой итерации, а не один раз перед **SCF** шагом, как если бы это было без нее. Такой расчет, безусловно, гораздо медленнее, но он позволяет не использовать дисковое пространство и операционную память под большое количество интегралов. Включение этой опции требуется при расчетах больших систем на компьютерах с маленьким диском и памятью.

Ghost-atoms Control (Использование атомов-призраков). Эта опция позволяет вводить центры, которым приписываются базисные функции тех или иных атомов, при этом в систему не вводятся ни дополнительные ядра, ни дополнительные электроны. Эта опция позволяет вводить только дополнительные базисные функции, центрированные в любой точке пространства. Необходимо помнить, что использование таких “виртуальных” атомов может приводить к неким артефактам. Например, так как существуют базисные волновые функции, будут существовать и мулликеновские заряды, которые будут центрированы на соответствующих центрах. Для того чтобы использовать эту возможность, необходимо в меню *Select*, предварительно сделав соответствующее выделение, выбрать пункт *Name selection* и, выбрав пункт *Other*, приписать ему имя **ghost-atoms**. После этого в диалоговом меню **Ab Initio Method / Advanced Options** становится активным пункт **Ghost-atoms Control**. Такой подход бывает эффективным при описании ряда неординарных химических связей. Эта опция активна только для расчетов одной точки.

MO initial guess (Параметр стартового заселения МО) определяет стартовое заполнение коэффициентов молекулярных орбиталей при помощи диагонализации остового гамильтониана. При выборе параметра **Projected Huckel**, эти параметры определяются по методу Хюккеля. Аналогично определяются и первоначальные коэффициенты при выборе параметров **Projected CNDO (методом CNDO)** и **Projected INDO (методом INDO)**.

Number of d Orbitals (Количество d-орбиталей). Этот параметр определяет вид *d*-орбиталей, используемых в расчете. Выбор **пяти (five)** орбиталей соответствует расчету с использованием эрмитовых орбиталей (*d* 0, *d* 1, *d* -1, *d* 2, *d* -2), а выбор **шести (six)** – соответствует расчету с использованием *d*-орбиталей в декартовом представлении (***d*_{xx}, *d*_{yy}, *d*_{zz}, *d*_{xy}, *d*_{xz}, *d*_{yz}**).

Кнопка *OK* служит для сохранения выбранных параметров и закрытия диалогового окна. Кнопка *Cancel* приводит к закрытию диалогового окна без сохранения выбранных параметров.

Меню **Compute** определяет вид расчета. Это может быть **расчет в одной точке (Single Point)**, **оптимизация геометрии (Geometry Optimization)**, **молекулярная динамика (Molecular Dynamics)**, **ланжевеновская динамика (Langevin Dynamics)**, **расчеты методом Монте-Карло**, а также некоторые другие виды расчетов. Далее по тексту книги нас в основном будут интересовать результаты расчетов методом молекулярной динамики, которые представлены файлами с динамическим кино. Для того чтобы просмотреть результаты **молекулярно-динамического моделирования**, необходимо прочитать соответствующий файл (в тексте они будут указываться), а затем, выбрав в меню **Compute** пункт **Molecular Dynamics**, отметить ☐ **Playback** и нажать **Proceed**. Для увеличения скорости проигрывания файлов (когда это необходимо, в зависимости от сложности задачи и мощности компьютера) можно увеличить **time steps** (по умолчанию стоит 1, при увеличении скорость увеличивается кратно количеству шагов).

Как уже говорилось выше, подробная инструкция о пользовании программой *HyperChem* дается в файлах *CDK.pdf*, *GetStart.pdf*, *Referenc.pdf*. Но научиться с ней работать не составляет большого труда, так как программа сконструирована очень удобно и предназначена для широкого круга пользователей с разным уровнем квалификации.

Для желающих более детально ознакомиться с некоторыми приемами и алгоритмами компьютерной химии, авторы могут порекомендовать хороший и легко доступный по пониманию источник [6], в котором они приведены.

Оглавление

Материалы к СРС и лабораторным работам	1
Использование мыши	3
Клавиатурные альтернативы.....	4
Кратчайшие клавиатурные пути.....	4
Открытие файла образца Sample File	4
Открытие HIN файла	4
Использование дисплейных установочных параметров	5
Использование подписей атомов и молекул	5
Использование молекулярных изображений	5
Основная техника построения и редактирования объектов	8
Рисование отдельных атомов:.....	8
Работа с выделенными атомами (молекулами).....	10
Выбор группы атомов.....	10
Удаление атомов	11
Копирование атома в буфер:.....	11
Очистка рабочей области <i>HyperChem</i> :.....	11
Создание полипептидов	12
Создание цвиттер-иона.....	12
Мутагенез.....	12
Упражнения.....	13
Измерение параметров структур (Measuring Structural Properties)	14
Характеристики атомов	14
Измерение длины связи.....	14
Измерение углов связей	14
Измерение торсионных углов	14
Измерение расстояния между двумя несвязанными атомами.....	15
Водородные связи	15
Создание небольших молекул в 2D и 3D (Упражнение)	15
Чтобы сохранить Вашу работу:	16
Маркирование атомов.....	16
Редактирование индивидуальных атомов	17
Расчеты в HyperChem	18
Молекулярная механика (ММ)	19
Диалоговое окно молекулярной механики MM+ Options.....	20
Диалоговое окно опций силового поля (Force Field Options Dialog Box)	22
Полуэмпирические методы расчета электронной структуры (Semi-empirical)	23
Диалоговое окно полуэмпирического метода.....	27
Диалоговое меню метода PMX.....	27
Диалоговое окно других полуэмпирических методов	28
Charge and Spin	28
Диалоговое окно Configuration Interaction.....	29
Расчеты в смешанном режиме (Квантовая механика/квантовая химия).....	31
Неэмпирический (ab initio) метод Хартри-Фока.....	31
Выбор базисного набора	32
Диалоговое окно метода ab initio.....	33
Диалоговое окно выбора других базисных наборов (Assign Other Basis Set Dialog Box)	34
Диалоговое окно параметров ab initio (Options)	34
SCF Controls (Секция параметров SCF расчета).....	35
Диалоговое окно параметров высокого уровня метода ab initio (Ab Initio Advanced options)	36

УДК 519.688

АНАЛИЗ ЭФФЕКТИВНОСТИ РЕШЕНИЯ ЗАДАЧИ N ТЕЛ НА РАЗЛИЧНЫХ ВЫЧИСЛИТЕЛЬНЫХ АРХИТЕКТУРАХ*

© 2009 г.

А.В. Адинец

Научно-исследовательский вычислительный центр
Московского государственного университета им. М.В. Ломоносова

adinetz@gmail.com

Поступила в редакцию 28.04.2009

Рассматриваются реализации решения задачи N тел на различных вычислительных архитектурах: графических процессорах AMD и NVidia, процессорах CELL, многоядерных процессорах, кластерных системах. Для реализации используются технологии C\$, SPU-C++, SSE, OpenMP, MPI, а также их комбинации. Для различных архитектур наблюдается различие производительности тривиальной и оптимизированной версий в 5 – 10 раз. Также показывается, что повышение производительности на разных архитектурах достигается использованием одних и тех же или сходных оптимизаций.

Ключевые слова: задача N тел, графические процессоры, параллельное программирование, вычислительные архитектуры.

Введение

Задача N тел является задачей расчета эволюции поведения системы из N тел, которые взаимодействуют при помощи некоторых сил [1]. Эта задача возникает при численном моделировании процессов в различных областях науки: молекулярной динамике, астрономии, жидкостной динамике, электростатике. При этом силы, действующие между частицами в модели, как правило, имеют дальнodelствующий характер. Если количество частиц N , то количество взаимодействий между ними, которые необходимо учитывать, растет как $O(N^2)$. Это обуславливает высокую вычислительную сложность задачи.

Помимо тривиального алгоритма взаимодействия, существуют различные приближенные схемы [2]. В среднем они позволяют снизить сложность с $O(N^2)$ до $O(N \log N)$ или даже $O(N)$, однако имеют худшие показатели точности. Не для всех имеются оценки ошибки сверху. Более того, для расчета взаимодействия в скоплениях близко расположенных частиц часть этих методов все равно использует взаимодействие типа «каждый с каждым» внутри скопления. А поскольку общее число частиц может быть достаточно большим, размер среднего скопления также оказывается большим. Что также требует

много вычислительных ресурсов для расчета $O(N^2)$ взаимодействий, где N в этом случае является размером скопления.

Таким образом, для решения задачи N тел в любом случае требуются большие вычислительные ресурсы. В настоящее время существует целый ряд высокопроизводительных вычислительных архитектур. Прежде всего, это традиционные центральные процессоры, а также построенные на их основе вычислительные кластеры. Далее идут системы на основе процессоров CELL BE [3], а также графические процессоры [4]. Наконец, для решения задачи N тел, взаимодействующих по гравитационному закону Ньютона, созданы специальные аппаратные решения, к примеру семейство систем GRAPE [5].

Данная статья рассматривает реализацию наиболее ресурсоемкой части задачи N тел – расчета взаимодействия частиц «каждый с каждым» – на различных современных вычислительных архитектурах. Для каждой архитектуры рассматривается несколько реализаций, от самой простой к более эффективной, с указанием используемых оптимизаций. Анализируются факторы, влияющие на производительность.

Задача N тел позволяет также оценить эффективность компилятора для данной архитектуры на вычислительно емких, но нетривиальных задачах. Для различных архитектур проводится сравнение эффективности решения на них данной задачи.

Данная статья организована следующим образом. В разделе 1 дается математическая фор-

* Статья рекомендована к печати программным комитетом Международной научной конференции «Параллельные вычислительные технологии 2009» (<http://agora.guru.ru/pavt>).

мулировка задачи N тел и краткий обзор методов ее решения. В разделе 2 приводится описание и сравнение современных вычислительных архитектур: графических процессоров, процессора CELL, а также обычных процессоров и кластерных систем на их основе. В разделе 3 для каждой из архитектур рассматриваются особенности реализации вычислительного ядра задачи N тел и оптимизации, применяемые программистом или компилятором. В разделе 4 приводятся результаты вычислительных экспериментов. При этом сравниваются оптимизированные и неоптимизированные версии программы для одной и той же архитектуры, обсуждается эффективность решения данной задачи. В заключении кратко суммируются основные результаты статьи.

1. Задача N тел

В задаче N тел взаимодействие вычисляется отдельно между парами частиц, а сила, действующая на каждую частицу (i -частицу) является суммой вкладов отдельных частиц (j -частиц), которые на нее действуют. Каждая частица характеризуется набором параметров, таких как масса или заряд. Кроме того, каждая частица характеризуется скоростью \mathbf{v}_i и положением \mathbf{r}_i . Задачей ядра вычисления взаимодействия является расчет ускорения, которое частица получает в результате влияния на нее других частиц:

$$\mathbf{a}_i = \frac{1}{m_i} \sum_{j=1, j \neq i}^N \mathbf{f}_{ij}, \quad (1)$$

где \mathbf{f}_{ij} — сила, с которой j -частица действует на i -частицу. При этом сила взаимодействия двух частиц вычисляется по некоторому закону и зависит, вообще говоря, от их взаимного положения, скорости, а также характеристик частиц. Примеры потенциалов взаимодействия приведены в [1]. Для гравитационного взаимодействия, на основании формулы (1), имеем:

$$\mathbf{a}_i = G \sum_{j=1, j \neq i}^N \frac{m_j}{|\mathbf{r}_j - \mathbf{r}_i|^3} (\mathbf{r}_j - \mathbf{r}_i). \quad (2)$$

Непосредственные вычисления по формуле (2) являются достаточно ресурсоемкими: для N частиц объем вычислений растет как $O(N^2)$. Поэтому помимо прямых методов для эффективного моделирования используются и более сложные схемы, позволяющие сократить объем вычислений.

Прямые схемы [6] для интегрирования обычно используют схему Эрмита, которая

требует вычисления не только ускорения, но и его производной по времени. Для каждой частицы поддерживается собственное значение времени и шага по времени, которое округляется до ближайшего значения на дискретной сетке, обычно экспоненциальной. В качестве шага по времени берется минимальный шаг из всех частиц. При этом положение и скорость обновляются только для тех частиц, для которых настало время, т.е. текущее время равно их шагу по времени плюс время частицы.

Древовидные схемы [7] позволяют сократить сложность вычислений до $O(N \log N)$. В этом случае влияние дальних групп тел вычисляется приближенно как взаимодействие с центром масс. Точность такой схемы оказывается ниже, и для некоторых реальных астрономических конфигураций ее использование может привести к неограниченным ошибкам в симуляции. Более того, при использовании данной схемы требуется вычислять взаимодействие между группами частиц, число которых может быть достаточно большим.

Мультипольные схемы [2] обладают ограниченной масштабируемостью при реализации на параллельных архитектурах. Сеточные методы, решающие уравнение Пуассона на сетке при помощи преобразования Фурье, требуют использования адаптивных сеток для достижения эффективности.

Анализ эффективности решения задачи N тел ранее проводился на различных вычислительных архитектурах. Рассматривались традиционные процессоры [8], кластеры на их основе [9] и графические процессоры [10, 11]. Публикации по использованию процессора CELL BE для решения гравитационной задачи N тел автору статьи неизвестны. Однако кластер Roadrunner на базе процессоров CELL использовался для моделирования межмолекулярного взаимодействия с короткодействующим потенциалом [12]. Помимо этого, автору неизвестны работы, где проводилось бы сравнение реализации задачи N тел на достаточно большом количестве архитектур.

2. Целевые архитектуры и средства программирования

2.1. Графические процессоры. Современные графические процессорные устройства (ГПУ) [4] в настоящее время активно используются для решения ресурсоемких задач. Соответствующее направление получило название *общих вычислений на ГПУ* (ОВГПУ, калька с англ. GPGPU – General Purpose GPU). Основными

причинами этого является их высокая производительность [13, 14] и высокая доступность, а также лучшие показатели энергетической эффективности и стоимости по сравнению с традиционными архитектурами.

В настоящее время существуют два основных производителя ГПУ для общих вычислений: компании AMD и NVidia. ГПУ этих производителей, хотя и отличаются в деталях, и используют различные технологии программирования, имеют сходную архитектуру. Общая схема архитектуры современных ГПУ и их места в видеокарте, плате приведена на рис. 1. Основными вычислительными устройствами ГПУ являются *поточковые процессоры (ПП)*. Их система команд включает арифметические операции с целыми и вещественными числами, а также команды асинхронного доступа к *видеоОЗУ*, ОЗУ ГПУ. При вычислениях на ГПУ все ПП исполняют одну и ту же вычислительную программу, называемую *шейдером*. Программа выполняется для всех узлов некоторой целочисленной решетки, размер которой может намного превосходить число ПП на ГПУ. Один запуск шейдера на такой целочисленной решетке называется *проходом*. Проход запускается под управлением *системы управления* ГПУ, которая отвечает за загрузку программы, распределение точек целочисленной решетки между ПП и т.д. На одних ГПУ все ПП имеют один и тот же счетчик команд, на других ГПУ они объединены в группы, имеющие общий счетчик команд. Таким образом, современные ГПУ представляют собой пример архитектуры типа ОКМД с общей памятью.

Проблема дисбаланса производительности процессора и памяти на ГПУ решается при по-

мощи сокрытия задержек с использованием *аппаратной многопоточности*. *Поток ГПУ* – это исполнение шейдера для одной точки целочисленной решетки. Если один из потоков заблокировался на доступ к памяти, *система управления* передает исполнение на другой поток. Поскольку контекст потока состоит только из регистров, переключение потоков осуществляется быстро.

Различия между архитектурами ГПУ разных производителей можно посмотреть, к примеру, в [15]. Там же приведен обзор средств программирования ГПУ. В качестве средства реализации на ГПУ задачи N тел выбрана система C\$ [16]. Она позволяет иметь общую базу кода, для различных ГПУ, кроме того, может автоматически оптимизировать код под конкретный ГПУ. Более того, одной из целей статьи является тестирование компиляторов для различных вычислительных платформ, в том числе и компилятора C\$.

2.2. Процессор CELL BE. Процессор CELL BE занимает промежуточное место между традиционными многоядерными процессорами и ГПУ. Общая схема его архитектуры приведена на рис. 2. Сам процессор состоит из набора главных и вспомогательных ядер. Главные ядра в основном выполняют управляющую функцию, хотя могут использоваться и для вычислений. Вспомогательные ядра выполняют только вычисления.

В роли главных ядер выступает PowerPC Processing Element (PPE, ППЭ – калька с английского). В роли вычислительных ядер выступают синергические процессорные элементы (СПЭ, англ. Synergistic Processing Element, SPE).

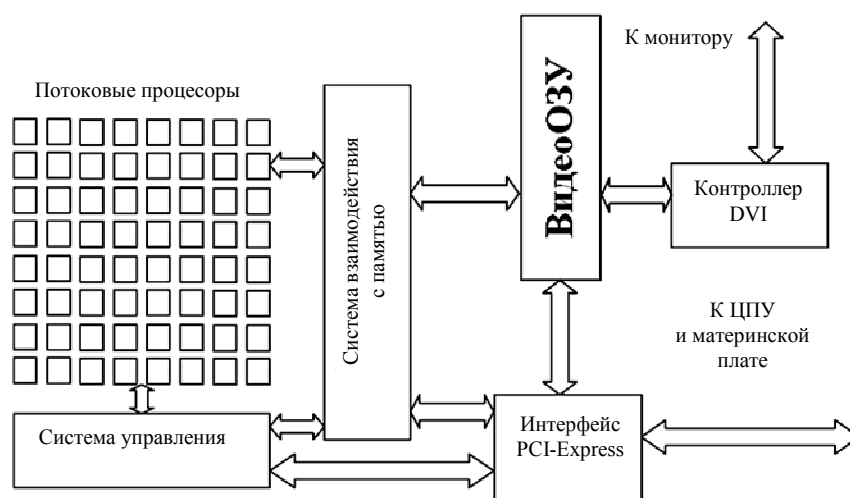


Рис.1. Общая схема современного ГПУ и его места в видеокарте и ПК

Каждый СПЭ состоит из синергического процессорного устройства (СПУ, англ. Synergistic Processing Unit, SPU) и контроллера потока памяти (КПП, англ. Memory Flow Controller, MFC). СПУ представляет собой векторный RISC-процессор, имеющий 128 регистров, каждый из которых имеет размер 128 бит. Каждый СПУ также имеет в своем распоряжении 256 кб явно адресуемой быстрой локальной памяти (ЛП).

СПУ не может напрямую обращаться к ОЗУ, это выполняется при помощи КПП. КПП является вспомогательным процессором и может

выполнять асинхронную передачу данных из ОЗУ объемами до 16 кб. Все элементы процессора CELL соединены внутренней шиной соединения элементов (англ. Element Interconnection Bus, EIB).

Для программирования CELL используется язык C/C++ с расширениями и библиотекой программирования, входящими в IBM CELL SDK [3]. В настоящее время необходимые расширения поддерживаются компиляторами gcc и IBM XLC/C++. Запуск программ на СПЭ осуществляется при помощи создания контекста СПЭ с загруженной в него программой, созда-



Рис. 1. Общая архитектура процессора CELL BE

Таблица 1

Сравнение характеристик тестовых систем

Архитектура	ГПУ AMD HD 4850	ГПУ Nvidia 8800GT	CELL BE	Intel Xeon E5472
Пиковая производительность, ГФлопс (одинарная/двойная точность)	800 / 160	340 / --	204 / 102	96 / 48
Количество ядер	160 (ПП)	128 (ПП)	8 (СПЭ)	4 (x86)
Регистров (на ядро)	до 128 x float4	до 64 x float	128 x float4	16 x float4 + обычные
Система команд ядра	VLIW/векторные трехадресные RISC	скалярные трехадресные RISC	векторные трехадресные RISC	векторные + скалярные двухадресные CISC
Память на чипе	L1: ~ 16 кбайт	16 кбайт явно адресуемой памяти на блок потоков	256 кбайт явно адресуемой памяти на СПУ	L1: 64 кбайт данные + 64 кбайт кода L2: 12 Мбайт
Пропускная способность канала процессор – ОЗУ	108 Гбайт/с	70 Гбайт/с	25 Гбайт/с	10 Гбайт/с
Доступ к ОЗУ	кэшируемый асинхронный	некэшируемый асинхронный	некэшируемый асинхронный	кэшируемый синхронный
Место в вычислительной системе	подчиненный (через PCI-E)	подчиненный (через PCI-E)	главный или подчиненный	главный
Мощность процессора/системы, Вт	160 / 500	145 / 400	- / 217	80 / 416 ¹
Энергетическая эффективность процессора/системы, МФлопс/Вт	5000 / 1600	2340 / 850	- / 1840 ²	600 / 230

¹В качестве системы рассматривается кластер «СКИФ-Чебышев», в дальнейшем используемый в вычислительных экспериментах.

²В качестве системы рассматривается IBM QS22 Blade Server.

```

// computes pairwise acceleration
public float3 accel(float mj, float3 ri, float3 rj) {
    float3 dr = rj - ri;
    float len2 = dr.len2;
    float3 a = mj / (len2 * sqrt(len2) + eps2) * dr;
    return a;
} // end of accel()

...
// computes acceleration for all particles
public void float3[] accelGpu() {
    a = let(i) G * + accel(ms[j], fr[i], fr[j]);
} // end of accelGpu()

```

Рис. 3. Фрагмент реализации задачи N тел на языке C\$

ния отдельного POSIX-потока и последующим исполнением в нем контекста СПЭ.

2.3. Обычные центральные процессоры и кластерные системы. В качестве «обычных» процессоров выступили четырехъядерные процессоры Intel Xeon E5472 (Harpertown) [17]. Из архитектурных особенностей следует отметить многоядерность, поддержку команд Streaming SIMD Extension (SSE) [18], а также кэш-память 1-го и 2-го уровней.

Для программирования использовался язык C++ и компилятор Intel C++. Многоядерность процессора задействовалась при помощи использования технологий OpenMP при работе на одном узле и MPI при работе на кластере. Для достижения максимальной производительности технология SSE задействовалась при помощи использования встроенных функций (англ. *intrinsics*) и специальных типов данных.

2.4. Сравнение различных архитектур. Сравнение характеристик различных архитектур приведено в табл. 1. ГПУ лидируют по показателям абсолютной производительности и энергетической эффективности процессора. Однако они отстают от процессора CELL по показателю энергетической эффективности системы в целом.

Также можно заметить сходство различных вычислительных архитектур. Все архитектуры, за исключением ГПУ NVidia, поддерживают векторные команды. В более новых процессорах используются трехадресные команды, в более старых – двухадресные. Для работы с векторными операндами используются регистры, содержащие четверки вещественных чисел. Заметим, что максимальная производительность на этих архитектурах может быть достигнута только за счет использования векторных команд.

Следует обратить внимание на иерархию памяти каждой из архитектур. ГПУ AMD и x64 используют кэш-память, в то время как в случае ГПУ NVidia и процессора CELL память на чипе является явно адресуемой. В процессоре CELL использование памяти на чипе обязательно, поскольку процессор не может получать доступ к данным ОЗУ напрямую. На ГПУ NVidia это необязательно, при этом программа, не использующая явно адресуемую память на чипе, будет работать медленно.

Все рассматриваемые архитектуры предоставляют возможность асинхронного доступа к ОЗУ. В случае ГПУ команды доступа к памяти являются асинхронными. В случае процессора CELL асинхронный доступ к ОЗУ осуществляется при помощи КПП, а в случае процессора x64 – при помощи команд предвыборки данных в кэш. Учет характера иерархии памяти в программе и грамотное использование асинхронного доступа позволяет повысить эффективность программы.

3. Особенности реализации алгоритма N тел для различных вычислительных архитектур

В данном разделе рассказывается об особенностях реализации алгоритма расчета ускорения по формуле (2). При этом расчет ускорения ведется в одинарной точности, поскольку расчеты с двойной точностью реализованы эффективно не на всех архитектурах.

3.1. Реализация для графических процессоров. Реализация для ГПУ выполнена с использованием языка C\$. С одной стороны, он позволяет создавать высокоуровневые программы, которые будут эффективно исполняться на ГПУ различных производителей [16]. С другой стороны, язык требует записи программы в высокоуровневых терминах, так что программист лишен возможности выполнять низкоуровневые

оптимизации, которые ему доступны, к примеру, в системе CUDA [19].

Фрагмент реализации задачи N тел на ГПУ на языке C# приведен на рис. 3. Поскольку программист в системе C# не имеет прямого контроля над низкоуровневыми оптимизациями, программа в системе запускалась один раз с включенными и один раз с выключенными оптимизациями. При включенных оптимизациях используется развертка редукций и векторизация. Для различных ГПУ задействовалась их иерархия памяти: для ГПУ AMD использовался большой регистровый файл, для ГПУ NVidia – статическая разделяемая память.

3.2. Реализация для ЦПУ, кластерных систем и CELL. Для обычных процессоров система создано приложение командной строки, реализующее расчеты по задаче N тел с использованием схемы Эйлера для интегрирования. Тип используемых оптимизаций, количество частиц и прочие настройки передаются как параметры командной строки.

Рассматриваются следующие оптимизации:

- Использование технологии SSE;
- Использование нескольких ядер при помощи технологии OpenMP;
- Использование нескольких узлов кластера при помощи технологии MPI.

Во избежание дублирования кода обычные и SSE-версии задачи N тел для обычных процессоров реализованы при помощи общего шаблона на C++.

Основным объектом программы является *решатель* (Solver) задачи N тел. Он выполняет загрузку и выгрузку данных из рабочего набора, а также последовательный вызов различных шагов процедуры решения.

За представление данных во время выполнения расчета отвечает класс *рабочего набора*. Он определяет используемые для представления данных типы и размещение данных в оперативной памяти. За обработку данных отвечают классы-*интеракторы*, задачей которых является обеспечение расчета взаимодействия i -частиц с j -частицами. Каждый интерактор может работать только с одним типом рабочего набора. С другой стороны, с каждым рабочим набором может быть связано несколько типов интеракторов. Например, с обычным локальным рабочим набором может быть связан как одноядерный интерактор, так и многоядерный интерактор, использующий технологию OpenMP. Рабочие наборы и интеракторы могут быть *локальными* и *распределенными*. Первые отвечают за вычисления на одном узле, вторые – за объеди-

нение узлов в кластер и взаимодействие между узлами.

Явное использование расширения SSE реализовано при помощи встроенных функций и типов (intrinsic). Чтобы ограничить зависимость кода от низкоуровневых особенностей архитектуры, работа с четверками вещественных чисел инкапсулирована в классе `qfloat`. Для него реализован набор стандартных арифметических операций и элементарных функций. Используемая в архитектуре схема рабочих наборов и интеракторов позволяет иметь один и тот же код для работы как с векторными командами, так и с обычными. Более того, такая схема позволяет легко инкапсулировать и использовать новые особенности архитектуры, например 8- или 16-элементные векторные операции, которые используются в процессорах Intel Nehalem и Intel Larrabee, соответственно.

При описанном выше подходе большая часть кода, специфичная для процессора CELL, уже присутствовала в готовом виде после выполнения реализации для процессоров x64.

Для СПУ потребовалось реализовать только отдельный интерактор и код, работающий на СПУ. Специальный код для CELL потребовался ввиду особенностей работы с памятью CELL. Помимо этого, один из внутренних циклов потребовалось развернуть вручную.

4. Результаты вычислительных экспериментов

4.1. Параметры эксперимента и тестовые системы. Основной задачей вычислительных экспериментов являлось определение эффективности реализации задачи N тел на различных вычислительных архитектурах, а также факторов, влияющих на эффективность. Для каждого значения архитектуры рассматриваются разные количества взаимодействующих частиц.

В литературе по задаче N тел принято считать, что количество вещественных операций при вычислении одного взаимодействия составляет 38, при этом дается ссылка на статью [20], где впервые появляется эта цифра. В самой статье указывается, что 38 – это число, которое получается при использовании конкретной схемы аппроксимации величины обратного квадратного корня на данной архитектуре; для других архитектур цифра может быть другой. На графических процессорах, к примеру, больше подойдет цифра 24, поскольку вычисление квадратного корня или обратной величины там имеет стоимость, равную стоимости 4 обычных операций. Для нашей текущей реализации на

CELL подходящая цифра была бы 32. В данной работе для оценки производительности считается, что вычисление одного взаимодействия занимает 24 операции, вне зависимости от используемой архитектуры.

В качестве тестовых ГПУ использовались AMD HD Radeon 4850 с пиковой производительностью 800 ГФлопс и NVidia GeForce 8800GTX с пиковой производительностью 340 ГФлопс. В качестве системы на базе процессора CELL BE выступал IBM QS22 Blade Server, содержащий 2 процессора CELL в качестве основных, всего 16 СПУ. Код для процессоров x64 тестировался на кластере «СКИФ-Чебышев», узлы которого содержат по 2 4-ядерных процессора Intel Xeon E5472 с пиковой производительностью 96 ГФлопс (с одинарной точностью). В зависимости от параметров программа запускалась на отдельном ядре, узле или множестве узлов.

4.2. Вычислительные эксперименты на ГПУ.

Результаты тестов на ГПУ даны в табл. 2. Эффективность, достигаемая на ГПУ AMD и NVidia при решении задачи N тел, позволяет говорить об эффективности используемых оптимизаций. Для ГПУ NVidia производительность C\$-программы близка к производительности программы, полученной в результате ручной оптимизации, которая составляет от 200–240 ГФлопс.

В качестве оптимизаций для ГПУ AMD система выполняет векторизацию цикла по i -частицам и развертку цикла суммирования взаимодействий по j -частицам. Глубина развертки внутреннего цикла составляет 16. Команды обращения в видеоОЗУ при этом переносятся в самое начало цикла. Для ГПУ NVidia система выполняет развертку цикла по j -частицам, а также задействует явно адресуемую память на чипе.

Включение оптимизаций C\$ на обоих типах ГПУ дает повышение производительности в 5–10 раз по сравнению с неоптимизированным вариантом. Объясняется это тем, что C\$ – достаточно высокоуровневый язык, который не дает пользователю доступа к низкоуровневым особенностям ГПУ. Поэтому их должна использовать система. Высокая достигаемая производительность свидетельствует, с другой стороны, об эффективности используемых оптимизаций для различных архитектур.

4.3. Вычислительные эксперименты на процессорах CELL. Результаты вычислительных экспериментов на процессоре CELL приведены

в таблице 3. Для каждого значения числа частиц указана производительность кода с векторизацией и без нее, на двух процессорах CELL (16 СПЭ). Векторизация дает значительный прирост производительности по сравнению с использованием обычного кода. Для большого числа частиц может достигаться более чем 20-кратное ускорение. Такое большое значение ускорения объясняется тем, что система команд СПЭ поддерживает *только векторные операции*. Таким образом, использование скалярных операций влечет за собой высокие накладные расходы, которые не возникают в коде, который использует векторизацию.

В случае использования векторизации задача является ограниченной производительностью АЛУ СПЭ, а не обменом данными с памятью. Использование двойной буферизации и другие оптимизации работы с памятью, обычно рекомендуемые для процессора CELL [3], не дают значительного роста производительности.

Данная задача практически идеально масштабируется по числу СПЭ. Этот результат вполне ожидаемый, поскольку на каждой итерации элементы целевого массива данных обрабатываются независимо.

4.4. Вычислительные эксперименты на обычных ЦПУ. На обычных ЦПУ отдельно проводились эксперименты на одном узле и на кластере. Результаты экспериментов на одном узле на одном ядре процессора приведены в табл. 4, на всем узле с использованием OpenMP – в табл. 5. В табл. 4 эффективность дается в расчете на одно ядро, в табл. 5 – на весь узел. Векторизация проводилась вручную, с использованием встроенных функций SSE. Использование векторизации дает ускорение в 3.6 раза, что близко к ожидаемому значению. Использование векторизации позволяет добиться примерно 50% эффективности решения задачи на ЦПУ.

Компилятор не сумел выполнить автоматическую векторизацию цикла, хотя внешний цикл не содержал зависимости по данным. Скорее всего, это объясняется сложностью используемого цикла: для достижения максимальной эффективности требуется выполнить векторизацию не только внутреннего цикла по j -частицам, но и внешнего.

Масштабируемость задачи на несколько ядер с использованием технологии OpenMP практически линейная, аналогично масштабируемости на большое число ядер в CELL.

На рис. 4 приведена зависимость ускорения выполнения задачи на кластере от числа используемых ядер. За основу берется произво-

Таблица 5

Результаты вычислительных экспериментов на ГПУ

ГПУ	Число частиц	Оптимизации	Производительность, ГФлопс	Эффективность, %
NVidia GeForce 8800GTX	4096	Нет	9.17	2.70
NVidia GeForce 8800GTX	32768	Нет	10.58	3.11
NVidia GeForce 8800GTX	32768	Да	209.93	61.74
NVidia GeForce 8800GTX	110592	Да	211.60	62.24
AMD Radeon HD4850	4096	Нет	113.24	14.15
AMD Radeon HD4850	32768	Нет	127.71	15.96
AMD Radeon HD4850	32768	Да	678.19	84.77
AMD Radeon HD4850	110592	Да	716.53	89.56

Таблица 5

Производительность решения задачи N тел на процессоре CELL

Число частиц	Векторизация	Производительность, ГФлопс	Эффективность, %
13824	Да	3.86	0.95
13824	Нет	75.38	18.48
32768	Да	3.90	0.96
32768	Нет	102.64	25.16
110592	Да	3.91	0.96
110592	Нет	109.13	26.75

Таблица 5

Результаты вычислительных экспериментов на x64 на одном ядре

Число частиц	Векторизация	Производительность	Эффективность, %
27000	Нет	3.25	13.56
27000	Да	11.75	48.97
64000	Нет	3.25	13.56
64000	Да	11.75	49.00

Таблица 5

Результаты вычислительных экспериментов на одном узле (8 ядер)

Число частиц	Векторизация	Производительность	Эффективность, %
64000	Нет	26.16386	13.63
64000	Да	94.77419	49.36
125000	Нет	26.15014	13.62
125000	Да	94.79059	49.37
216000	Нет	26.15442	13.62
216000	Да	94.83355	49.39

дительность задачи на одном узле, т.е. на 8 ядрах. Масштабируемость снова линейная, скорость выполнения ограничена производительностью ЦПУ, а не коммуникационной среды. Эффективность решения задачи на кластере примерно такая же, как и на одном узле, и составляет 50%.

Между отдельными запусками задачи, однако, может наблюдаться разброс по времени. Связано это с используемой схемой реализации, которая чувствительна к однородности скорости работы процессоров. То есть если в системе по каким-то причинам один из процессоров оказывается «медленным», то время решения задачи становится таким же, как если бы ее решал кластер, состоящий только из таких «медленных» процессоров. Причинами замедления

могут быть неравномерность тактовой частоты, системный шум, другие задачи, работающие на том же процессоре, и т.д. При этом по мере увеличения числа используемых узлов вероятность попадания на «медленный» процессор возрастает.

4.5. Сравнение эффективности реализаций.

В табл. 6 сравнивается эффективность работы задачи N тел на различных архитектурах. Можно видеть, что задача N тел с одинарной точностью наиболее эффективно решается с использованием графических процессоров. На данной задаче достигается от 60 до 90% пиковой производительности ГПУ, а реально достигаемая энергетическая эффективность достигает 1.4 ГФлопс/Вт. При этом энергетическая эф-

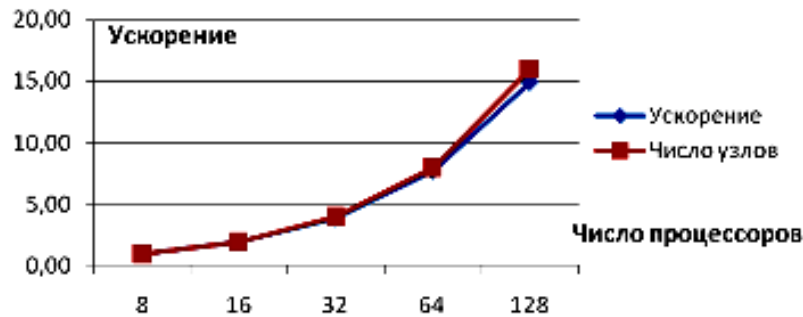
Рис. 4. Масштабируемость задачи N тел на кластере

Таблица 6

Эффективность решения задачи N тел на различных архитектурах

Архитектура	Достигнутая производительность, ГФлопс	Пиковая производительность, ГФлопс	Эффективность, %	Энергетическая эффективность, МФлопс/Вт
AMD HD4850	716.53	800	89.56	1433
NVidia 8800GTX	211.60	340	62.24	529
QS22 Blade Server	109.13	408	26.75	503
«СКИФ-Чебышев», 1 узел	94.83	192	49.39	114
«СКИФ-Чебышев», 16 узлов	1410.97	3072	45.93	106

фективность рассчитывается для всей системы, в которой большая часть энергии потребляется не ГПУ, а вспомогательным оборудованием — ЦПУ, материнской платой и т.д. По нашим оценкам, установка нескольких графических карт в одну машину позволит повысить энергетическую эффективность примерно в 2.5 раза.

Наиболее низкая загруженность достигается на процессоре CELL. Скорее всего, это связано с неэффективной работой компилятора `spu-g++` даже при самых сильных настройках оптимизации. Достигаемая при этом производительность лишь ненамного превосходит ту, что достигается на «обычном» $x64$ -процессоре. По нашим оценкам, однако, имеется возможность повысить эффективность примерно в 1.5 раза и достигнуть загруженности в 40%. По показателям энергетической эффективности CELL находится между ГПУ и обычными процессорами. При этом возможности сокращения потребления для CELL по сравнению с рассматриваемыми системами на основе ГПУ ограничены, поскольку QS22 Blade Server изначально создавался с целью минимизации энергопотребления, а поэтому большая часть возможностей уже использована.

Обычные процессоры, хотя и демонстрируют эффективность около 50%, сильно проигрывают нетрадиционным архитектурам по энергетической эффективности. По этому показателю они примерно в 4.5 раза хуже CELL и более чем на порядок уступают графическим процессорам.

При этом следует заметить, что традиционные компиляторы не могут эффективно транслировать гнездо из 2 циклов, которое используется в задаче N тел, несмотря на отсутствие в нем зависимостей. Система C\$ более эффективно справляется с этим гнездом циклов. Достигается это за счет использования альтернативного подхода: программа рассматривается не как цикл, а как набор операций с массивами и функциями, при этом осуществляется поиск наиболее эффективного способа его отображения на целевую архитектуру. Массивы при этом доступны только на чтение или однократную запись. Кроме того, массивы рассматриваются как абстрактные типы данных, для которых способ представления на конкретной архитектуре может выбираться в зависимости от особенностей решаемой задачи. На других архитектурах более эффективные способы представления и отображения реализуются вручную.

Графические процессоры наиболее эффективны для решения задачи N тел только при использовании одинарной точности. Если для расчетов взаимодействия потребуется использовать двойную точность, показатели их энергетической эффективности снизятся примерно в 5 раз. В этом случае наиболее эффективной архитектурой для данной задачи будет процессор CELL; однако ГПУ все равно будут более эффективными, чем обычные процессоры.

На всех архитектурах производительность оптимизированной и неоптимизированной версий отличается не менее чем в несколько раз, а иногда и на порядки. При этом везде используются одинаковый набор оптимизаций:

- Векторизация (если есть векторные команды);
- Развертка цикла;
- Использование иерархии памяти.

Рассматриваемые в данной статье подходы к решению задачи N тел позволяют отметить ряд способов повышения эффективности программирования вычислительно емких задач.

- Использование более сложных оптимизаций в компиляторе. Используемые в C# оптимизации могут быть перенесены и на другие вычислительные архитектуры. Их использование может существенно увеличить время компиляции; эффективным решением может быть использование директив компилятора, которые включали бы такую оптимизацию только для определенных участка кода.

- Использование метапрограммирования в сочетании со специальными типами для работы с векторными типами и размещением данных в ОЗУ. Данный подход требует больших усилий со стороны программиста. С другой стороны, он дает ему большие возможности оптимизации для конкретной архитектуры. Активное использование шаблонов и ООП позволит избежать дублирования кода и появления большого количества кода, работающего только на одной архитектуре. Однако сам код при этом становится менее читаемым.

- Использование систем автоматической генерации и преобразования исходных кодов программ.

Заключение

В статье рассмотрены реализации вычислительного ядра задачи N тел на ряде современных вычислительных архитектур: процессоров x64, кластеров на основе процессоров x64, процессоров CELL, графических процессоров. Для каждой из архитектур рассматривается несколько возможных реализаций, начиная с простых и неэффективных и заканчивая реализациями с высокой эффективностью. Для архитектур исследовано влияние различных оптимизаций на эффективность выполнения. При этом показано, что исследуемые архитектуры обладают множеством сходных признаков, и как следствие, для повышения эффективности программ на них используются сходные оптимизации.

Также показано, что разница в производительности простых и оптимизированных программ может быть в несколько раз, а на некоторых архитектурах – на порядок. При этом оптимизацию часто требуется выполнять вручную, автоматическая векторизация или развертка циклов у современных компиляторов работает только в самых простых случаях.

На примере задачи N тел проведено сравнение различных архитектур. Показано, что для данной задачи, как и для большинства вычислительно емких задач, нетрадиционные архитектуры превосходят обычные процессоры по таким характеристикам, как производительность и энергетическая эффективность. При этом сложность достижения высокой производительности как на тех, так и на других может требовать длительной ручной оптимизации. В этих условиях наличие общего языка программирования позволяет иметь единый код для нескольких архитектур, сокращая расходы на адаптацию и оптимизацию.

Список литературы

1. Elsen E. et al. N-Body Simulations on GPUs // Proceedings of ACM/IEEE Conference on Supercomputing, 2006.
2. N-body simulations // Scholarpedia. URL: http://www.scholarpedia.org/article/N-body_simulations
3. IBM Cell Broadband Engine Resource Center. URL: <http://www.ibm.com/developerworks/power/cell/>
4. GPGPU.org. URL: <http://www.gpgpu.org>
5. Makino J., Kokubo E., Fukushima T. Performance evaluation and tuning of GRAPE-6 – towards 40 «real» Tflops // Proceedings of ACM/IEEE conference on Supercomputing-2003.
6. Makino J., Aarseth S. J. On a Hermite integrator with Ahmad-Cohen scheme for gravitational many-body problems // PASJ: Publications of the Astronomical Society of Japan. 1992. V. 44. P. 141–151.
7. Barnes J., Hut P. A hierarchical $O(N \log N)$ force-calculation algorithm // Nature. 1986. V. 324. P. 446–449.
8. Nitadori K., Makino J., Hut P. Performance Tuning of N-Body Codes on Modern Microprocessors: I. Direct Integration with a Hermite Scheme on x86_64 Architecture. URL: <http://arxiv.org/abs/astro-ph/0511062>.
9. Gualandris A., Zwart S. P., Tirado-Ramos A. Performance analysis of direct N-body algorithms for astrophysical simulations on distributed systems. URL: <http://arxiv.org/abs/astro-ph/0412206>.
10. Belleman R., Bedorf J., Zwart S. High performance direct gravitational N-body simulations on graphics processing units II: An implementation in CUDA. URL: <http://arxiv.org/abs/0707.0438v2>.
11. Hamada T., Iitaka T. The Chamomile Scheme: an optimized algorithm for N-body simulations on

- programmable graphics processing units. URL: <http://arxiv.org/abs/astro-ph/0703100>.
12. Swaminarayan S. et al. 369 Tflo/s molecular dynamics simulations on the Roadrunner general-purpose heterogeneous supercomputer // Proceedings of ACM/IEEE Conference on Supercomputing-2008.
13. AMD FireStream 9270 // AMD ATI site. URL: http://ati.amd.com/technology/streamcomputing/product_firestream_9270.html.
14. NVidia Tesla C1060. URL: http://www.nvidia.com/object/product_tesla_c1060_us.html.
15. Адинец А., Воеводин Вл.В. Графический вызов суперкомпьютерам // Открытые системы. 2008. № 4. С. 35–41.
16. Адинец А.В., Кривов М.А. Методы оптимизации программ для современных графических процессоров // Труды Всероссийской научной конференции «Научный сервис в сети Интернет-2008: решение больших задач». С. 345–353.
17. Процессоры Intel Xeon серий 5xxx // Информационно-аналитический центр parallel.ru. URL: <http://parallel.ru/russia/MSU-Intel/xeons.html>.
18. Intel Corp. Intel SSE4 Programming Reference. URL: <http://softwarecommunity.intel.com/isn/Downloads/Intel%20SSE4%20Programming%20Reference.pdf>
19. NVidia CUDA Programming Guide v1.1. URL: http://developer.download.nvidia.com/compute/cuda/1_1/NVIDIA_CUDA_Programming_Guide_1.1.pdf
20. Warren M. et al. Pentium Pro Inside: I. A Treecode at 430 Gigafllops on ASCI Red, II. Price/Performance of \$50/Mflop on Loki and Hyglac // Proceedings of SuperComputing 1997.

ANALYZING THE EFFICIENCY OF N-BODY SIMULATIONS ON DIFFERENT COMPUTATIONAL ARCHITECTURES

A.V. Adinetz

N-body simulation is the process of modeling the behavior of a system of particles with long-distance interactions between them. Different variants of such simulation are used in astronomy and molecular dynamics, where particles interact by means of gravitational and molecular forces, respectively. N-body simulation has a very large computational complexity, because $O(N^2)$ interactions needs to be computed for a system of N particles at each simulation step. Parallel computations and supercomputers are therefore required for such simulation. In this paper, a variety of architectures are used for N-body simulations: AMD and NVidia graphics processors, CELL and multi-core processors, computer clusters. Programming tools used are C\$, C++, SPU-C++, SSE, OpenMP, MPI and their combinations. For all architectures, the difference of 5-10 times between a trivial and an optimized version is observed. It is also demonstrated that similar optimizations can be used on different architectures in order to increase performance.

Keywords: N-body simulation, graphics processors, GPU, parallel programming, computational architectures.



Chapter 31

Fast N-Body Simulation with CUDA

Lars Nyland
NVIDIA Corporation

Mark Harris
NVIDIA Corporation

Jan Prins
University of North Carolina at Chapel Hill

31.1 Introduction

An N-body simulation numerically approximates the evolution of a system of bodies in which each body continuously interacts with every other body. A familiar example is an astrophysical simulation in which each body represents a galaxy or an individual star, and the bodies attract each other through the gravitational force, as in Figure 31-1. N-body simulation arises in many other computational science problems as well. For example, protein folding is studied using N-body simulation to calculate electrostatic and van der Waals forces. Turbulent fluid flow simulation and global illumination computation in computer graphics are other examples of problems that use N-body simulation.

The *all-pairs* approach to N-body simulation is a brute-force technique that evaluates all pair-wise interactions among the N bodies. It is a relatively simple method, but one that is not generally used on its own in the simulation of large systems because of its $O(N^2)$ computational complexity. Instead, the all-pairs approach is typically used as a kernel to determine the forces in close-range interactions. The all-pairs method is combined with a faster method based on a far-field approximation of longer-range forces, which is valid only between parts of the system that are well separated. Fast N-body

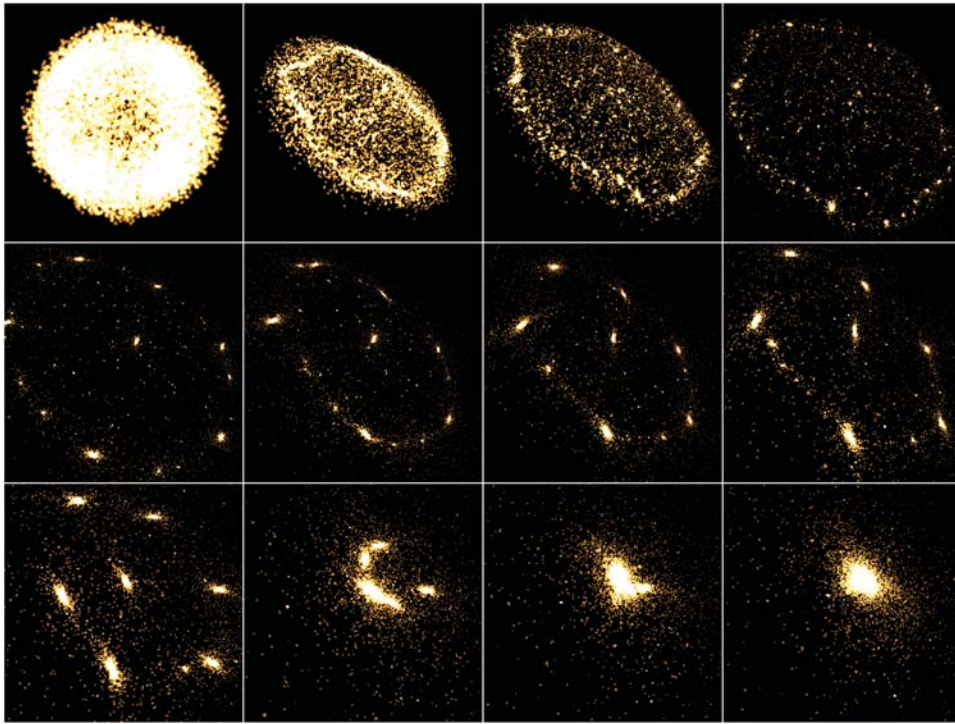


Figure 31-1. Frames from an Interactive 3D Rendering of a 16,384-Body System Simulated by Our Application

We compute more than 10 billion gravitational forces per second on an NVIDIA GeForce 8800 GTX GPU, which is more than 50 times the performance of a highly tuned CPU implementation.

algorithms of this form include the Barnes-Hut method (BH) (Barnes and Hut 1986), the fast multipole method (FMM) (Greengard 1987), and the particle-mesh methods (Hockney and Eastwood 1981, Darden et al. 1993).

The all-pairs component of the algorithms just mentioned requires substantial time to compute and is therefore an interesting target for acceleration. Improving the performance of the all-pairs component will also improve the performance of the far-field component as well, because the balance between far-field and near-field (all-pairs) can be shifted to assign more work to a faster all-pairs component. Accelerating one component will offload work from the other components, so the entire application benefits from accelerating one kernel.

In this chapter, we focus on the all-pairs computational kernel and its implementation using the NVIDIA CUDA programming model. We show how the parallelism available in the all-pairs computational kernel can be expressed in the CUDA model and how various parameters can be chosen to effectively engage the full resources of the NVIDIA GeForce 8800 GTX GPU. We report on the performance of the all-pairs N-body kernel for astrophysical simulations, demonstrating several optimizations that improve performance. For this problem, the GeForce 8800 GTX calculates more than 10 billion interactions per second with $N = 16,384$, performing 38 integration time steps per second. At 20 flops per interaction, this corresponds to a sustained performance in excess of 200 gigaflops. This result is close to the theoretical peak performance of the GeForce 8800 GTX GPU.

31.2 All-Pairs N-Body Simulation

We use the gravitational potential to illustrate the basic form of computation in an all-pairs N-body simulation. In the following computation, we use bold font to signify vectors (typically in 3D). Given N bodies with an initial position \mathbf{x}_i and velocity \mathbf{v}_i for $1 \leq i \leq N$, the force vector \mathbf{f}_{ij} on body i caused by its gravitational attraction to body j is given by the following:

$$\mathbf{f}_{ij} = G \frac{m_i m_j}{\|\mathbf{r}_{ij}\|^2} \cdot \frac{\mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|},$$

where m_i and m_j are the masses of bodies i and j , respectively; $\mathbf{r}_{ij} = \mathbf{x}_j - \mathbf{x}_i$ is the vector from body i to body j ; and G is the gravitational constant. The left factor, the *magnitude* of the force, is proportional to the product of the masses and diminishes with the square of the distance between bodies i and j . The right factor is the *direction* of the force, a unit vector from body i in the direction of body j (because gravitation is an attractive force).

The total force \mathbf{F}_i on body i , due to its interactions with the other $N - 1$ bodies, is obtained by summing all interactions:

$$\mathbf{F}_i = \sum_{\substack{1 \leq j \leq N \\ j \neq i}} \mathbf{f}_{ij} = G m_i \cdot \sum_{\substack{1 \leq j \leq N \\ j \neq i}} \frac{m_j \mathbf{r}_{ij}}{\|\mathbf{r}_{ij}\|^3}.$$

As bodies approach each other, the force between them grows without bound, which is an undesirable situation for numerical integration. In astrophysical simulations, collisions between bodies are generally precluded; this is reasonable if the bodies represent galaxies that may pass right through each other. Therefore, a *softening factor* $\varepsilon^2 > 0$ is added, and the denominator is rewritten as follows:

$$\mathbf{F}_i \approx Gm_i \cdot \sum_{1 \leq j \leq N} \frac{m_j \mathbf{r}_{ij}}{\left(\|\mathbf{r}_{ij}\|^2 + \varepsilon^2\right)^{3/2}}.$$

Note the condition $j \neq i$ is no longer needed in the sum, because $\mathbf{f}_{ii} = 0$ when $\varepsilon^2 > 0$. The softening factor models the interaction between two Plummer point masses: masses that behave as if they were spherical galaxies (Aarseth 2003, Dyer and Ip 1993). In effect, the softening factor limits the magnitude of the force between the bodies, which is desirable for numerical integration of the system state.

To integrate over time, we need the acceleration $\mathbf{a}_i = \mathbf{F}_i/m_i$ to update the position and velocity of body i , and so we simplify the computation to this:

$$\mathbf{a}_i \approx G \cdot \sum_{1 \leq j \leq N} \frac{m_j \mathbf{r}_{ij}}{\left(\|\mathbf{r}_{ij}\|^2 + \varepsilon^2\right)^{3/2}}.$$

The integrator used to update the positions and velocities is a leapfrog-Verlet integrator (Verlet 1967) because it is applicable to this problem and is computationally efficient (it has a high ratio of accuracy to computational cost). The choice of integration method in N-body problems usually depends on the nature of the system being studied. The integrator is included in our timings, but discussion of its implementation is omitted because its complexity is $O(N)$ and its cost becomes insignificant as N grows.

31.3 A CUDA Implementation of the All-Pairs N-Body Algorithm

We may think of the all-pairs algorithm as calculating each entry \mathbf{f}_{ij} in an $N \times N$ grid of all pair-wise forces.¹ Then the total force \mathbf{F}_i (or acceleration \mathbf{a}_i) on body i is obtained from the sum of all entries in row i . Each entry can be computed independently, so there is $O(N^2)$ available parallelism. However, this approach requires $O(N^2)$ memory

1. The relation between reciprocal forces $\mathbf{f}_{ji} = -\mathbf{f}_{ij}$ can be used to reduce the number of force evaluations by a factor of two, but this optimization has an adverse effect on parallel evaluation strategies (especially with small N), so it is not employed in our implementation.

and would be substantially limited by memory bandwidth. Instead, we serialize some of the computations to achieve the data reuse needed to reach peak performance of the arithmetic units and to reduce the memory bandwidth required.

Consequently, we introduce the notion of a computational *tile*, a square region of the grid of pair-wise forces consisting of p rows and p columns. Only $2p$ body descriptions are required to evaluate all p^2 interactions in the tile (p of which can be reused later). These body descriptions can be stored in shared memory or in registers. The total effect of the interactions in the tile on the p bodies is captured as an update to p acceleration vectors.

To achieve optimal reuse of data, we arrange the computation of a tile so that the interactions in each row are evaluated in sequential order, updating the acceleration vector, while the separate rows are evaluated in parallel. In Figure 31-2, the diagram on the left shows the evaluation strategy, and the diagram on the right shows the inputs and outputs for a tile computation.

In the remainder of this section, we follow a bottom-up presentation of the full computation, packaging the available parallelism and utilizing the appropriate local memory at each level.

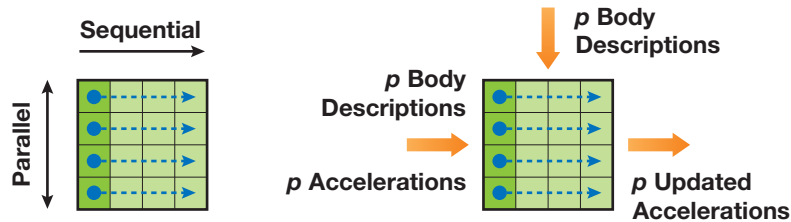


Figure 31-2. A Schematic Figure of a Computational Tile

Left: Evaluation order. Right: Inputs needed and outputs produced for the p^2 interactions calculated in the tile.

31.3.1 Body-Body Force Calculation

The interaction between a pair of bodies as described in Section 31.2 is implemented as an entirely serial computation. The code in Listing 31-1 computes the force on body i from its interaction with body j and updates acceleration \mathbf{a}_i of body i as a result of this interaction. There are 20 floating-point operations in this code, counting the additions, multiplications, the `sqrtf()` call, and the division (or reciprocal).

Listing 31-1. Updating Acceleration of One Body as a Result of Its Interaction with Another Body

```
__device__ float3
bodyBodyInteraction(float4 bi, float4 bj, float3 ai)
{
    float3 r;

    // r_ij [3 FLOPS]
    r.x = bj.x - bi.x;
    r.y = bj.y - bi.y;
    r.z = bj.z - bi.z;

    // distSqr = dot(r_ij, r_ij) + EPS^2 [6 FLOPS]
    float distSqr = r.x * r.x + r.y * r.y + r.z * r.z + EPS2;

    // invDistCube = 1/distSqr^(3/2) [4 FLOPS (2 mul, 1 sqrt, 1 inv)]
    float distSixth = distSqr * distSqr * distSqr;
    float invDistCube = 1.0f/sqrtf(distSixth);

    // s = m_j * invDistCube [1 FLOP]
    float s = bj.w * invDistCube;

    // a_i = a_i + s * r_ij [6 FLOPS]
    ai.x += r.x * s;
    ai.y += r.y * s;
    ai.z += r.z * s;

    return ai;
}
```

We use CUDA's `float4` data type for body descriptions and accelerations stored in GPU device memory. We store each body's mass in the *w* field of the body's `float4` position. Using `float4` (instead of `float3`) data allows *coalesced* memory access to the arrays of data in device memory, resulting in efficient memory requests and transfers. (See the *CUDA Programming Guide* (NVIDIA 2007) for details on coalescing memory requests.) Three-dimensional vectors stored in local variables are stored as `float3` variables, because register space is an issue and coalesced access is not.

31.3.2 Tile Calculation

A tile is evaluated by *p* threads performing the same sequence of operations on different data. Each thread updates the acceleration of one body as a result of its interaction with

p other bodies. We load p body descriptions from the GPU device memory into the shared memory provided to each *thread block* in the CUDA model. Each thread in the block evaluates p successive interactions. The result of the tile calculation is p updated accelerations.

The code for the tile calculation is shown in Listing 31-2. The input parameter `myPosition` holds the position of the body for the executing thread, and the array `shPosition` is an array of body descriptions in shared memory. Recall that p threads execute the function body in parallel, and each thread iterates over the same p bodies, computing the acceleration of its individual body as a result of interaction with p other bodies.

Listing 31-2. Evaluating Interactions in a $p \times p$ Tile

```
__device__ float3
tile_calculation(float4 myPosition, float3 accel)
{
    int i;
    extern __shared__ float4[] shPosition;

    for (i = 0; i < blockDim.x; i++) {
        accel = bodyBodyInteraction(myPosition, shPosition[i], accel);
    }
    return accel;
}
```

The G80 GPU architecture supports concurrent reads from multiple threads to a single shared memory address, so there are no shared-memory-bank conflicts in the evaluation of interactions. (Refer to the *CUDA Programming Guide* (NVIDIA 2007) for details on the shared memory broadcast mechanism used here.)

31.3.3 Clustering Tiles into Thread Blocks

We define a thread block as having p threads that execute some number of tiles in sequence. Tiles are sized to balance parallelism with data reuse. The degree of parallelism (that is, the number of rows) must be sufficiently large so that multiple warps can be interleaved to hide latencies in the evaluation of interactions. The amount of data reuse grows with the number of columns, and this parameter also governs the size of the transfer of bodies from device memory into shared memory. Finally, the size of the tile also determines the register space and shared memory required. For this implementation, we

have used square tiles of size p by p . Before executing a tile, each thread fetches one body into shared memory, after which the threads synchronize. Consequently, each tile starts with p successive bodies in the shared memory.

Figure 31-3 shows a thread block that is executing code for multiple tiles. Time spans the horizontal direction, while parallelism spans the vertical direction. The heavy lines demarcate the tiles of computation, showing where shared memory is loaded and a barrier synchronization is performed. In a thread block, there are N/p tiles, with p threads computing the forces on p bodies (one thread per body). Each thread computes all N interactions for one body.

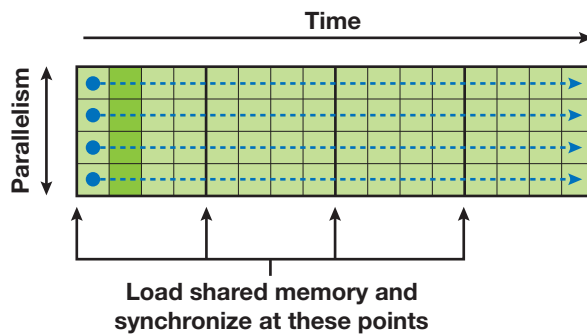


Figure 31-3. The CUDA Kernel of Pair-Wise Forces to Calculate
Multiple threads work from left to right, synchronizing at the end of each tile of computation.

The code to calculate N-body forces for a thread block is shown in Listing 31-3. This code is the CUDA kernel that is called from the host.

The parameters to the function `calculate_forces()` are pointers to global device memory for the positions `devX` and the accelerations `devA` of the bodies. We assign them to local pointers with type conversion so they can be indexed as arrays. The loop over the tiles requires two synchronization points. The first synchronization ensures that all shared memory locations are populated before the gravitation computation proceeds, and the second ensures that all threads finish their gravitation computation before advancing to the next tile. Without the second synchronization, threads that finish their part in the tile calculation might overwrite the shared memory still being read by other threads.

Listing 31-3. The CUDA Kernel Executed by a Thread Block with p Threads to Compute the Gravitational Acceleration for p Bodies as a Result of All N Interactions

```
__global__ void
calculate_forces(void *devX, void *devA)
{
    extern __shared__ float4[] shPosition;

    float4 *globalX = (float4 *)devX;
    float4 *globalA = (float4 *)devA;
    float4 myPosition;
    int i, tile;
    float3 acc = {0.0f, 0.0f, 0.0f};
    int gtid = blockIdx.x * blockDim.x + threadIdx.x;

    myPosition = globalX[gtid];

    for (i = 0, tile = 0; i < N; i += p, tile++) {
        int idx = tile * blockDim.x + threadIdx.x;
        shPosition[threadIdx.x] = globalX[idx];
        __syncthreads();
        acc = tile_calculation(myPosition, acc);
        __syncthreads();
    }
    // Save the result in global memory for the integration step.
    float4 acc4 = {acc.x, acc.y, acc.z, 0.0f};
    globalA[gtid] = acc4;
}
```

31.3.4 Defining a Grid of Thread Blocks

The kernel program in Listing 31-3 calculates the acceleration of p bodies in a system, caused by their interaction with all N bodies in the system. We invoke this program on a *grid* of thread blocks to compute the acceleration of all N bodies. Because there are p threads per block and one thread per body, the number of thread blocks needed to complete all N bodies is N/p , so we define a 1D grid of size N/p . The result is a total of N threads that perform N force calculations each, for a total of N^2 interactions.

Evaluation of the full grid of interactions can be visualized as shown in Figure 31-4. The vertical dimension shows the parallelism of the 1D grid of N/p independent

thread blocks with p threads each. The horizontal dimension shows the sequential processing of N force calculations in each thread. A thread block reloads its shared memory every p steps to share p positions of data.

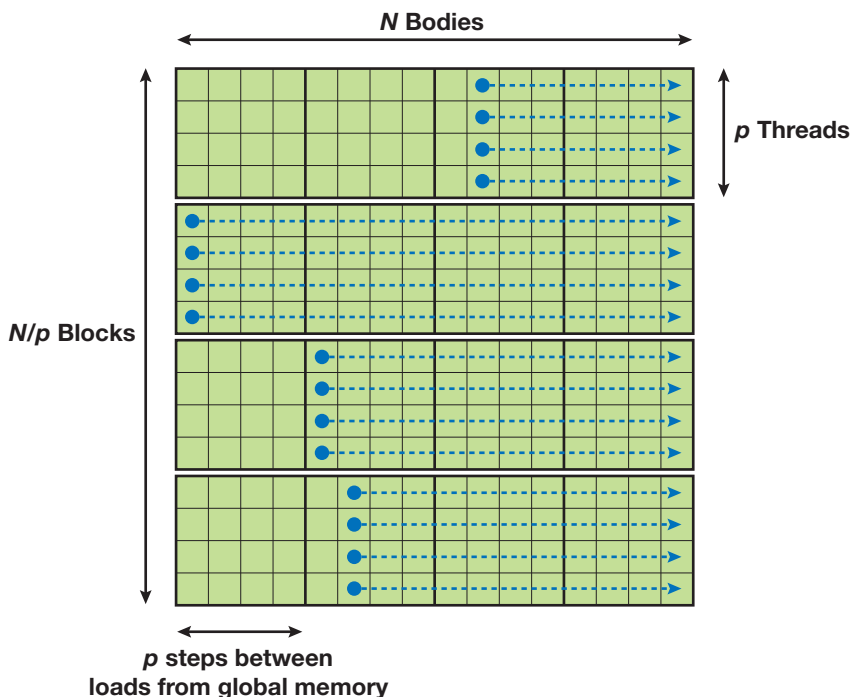


Figure 31-4. The Grid of Thread Blocks That Calculates All N^2 Forces
Here there are four thread blocks with four threads each.

31.4 Performance Results

By simply looking at the clocks and capacities of the GeForce 8800 GTX GPU, we observe that it is capable of 172.8 gigaflops (128 processors, 1.35 GHz each, one floating-point operation completed per cycle per processor). Multiply-add instructions (MADs) perform two floating-point operations every clock cycle, doubling the potential performance. Fortunately, the N-body code has several instances where MAD instructions are generated by the compiler, raising the performance ceiling well over 172.8 gigaflops.

Conversely, complex instructions such as inverse square root require multiple clock cycles. The *CUDA Programming Guide* (NVIDIA 2007) says to expect 16 clock cycles

per warp of 32 threads, or four times the amount of time required for the simpler operations. Our code uses one inverse-square-root instruction per interaction.

When comparing gigaflop rates, we simply count the floating-point operations listed in the high-level code. By counting the floating-point operations in the `bodyBody-Interaction` code (Listing 31-1), we see nine additions, nine multiplications, one square root, and one division. Division and square root clearly require more time than addition or multiplication, and yet we still assign a cost of 1 flop each,² yielding a total of 20 floating-point operations per pair-wise force calculation. This value is used throughout the chapter to compute gigaflops from interactions per second.

31.4.1 Optimization

Our first implementation achieved 163 gigaflops for 16,384 bodies. This is an excellent result, but there are some optimizations we can use that will increase the performance.

Performance Increase with Loop Unrolling

The first improvement comes from *loop unrolling*, where we replace a single body-body interaction call in the inner loop with 2 to 32 calls to reduce loop overhead. A chart of performance for small unrolling factors is shown in Figure 31-5.

We examined the code generated by the CUDA compiler for code unrolled with 4 successive calls to the body-body interaction function. It contains 60 instructions for the 4 in-lined calls. Of the 60 instructions, 56 are floating-point instructions, containing 20 multiply-add instructions and 4 inverse-square-root instructions. Our best hope is that the loop will require 52 cycles for the non-inverse-square-root floating-point instructions, 16 cycles for the 4 inverse-square-root instructions, and 4 cycles for the loop control, totaling 72 cycles to perform 80 floating-point operations.

If this performance is achieved, the G80 GPU will perform approximately 10 billion body-body interactions per second (128 processors at 1350 MHz, computing 4 body-body interactions in 72 clock cycles), or more than 200 gigaflops. This is indeed the performance we observe for $N > 8192$, as shown in Figure 31-6.

2. Although we count $1.0/\sqrt{x}$ as two floating-point operations, it may also be assumed to be a single operation called “`rsqrt()`” (Elsen et al. 2006). Doing so reduces the flop count per interaction to 19 instead of 20. Some researchers use a flop count of 38 for the interaction (Hamada and Iitaka 2007); this is an arbitrary conversion based on an historical estimate of the running time equivalent in flops of square root and reciprocal. It bears no relation to the actual number of floating-point operations.

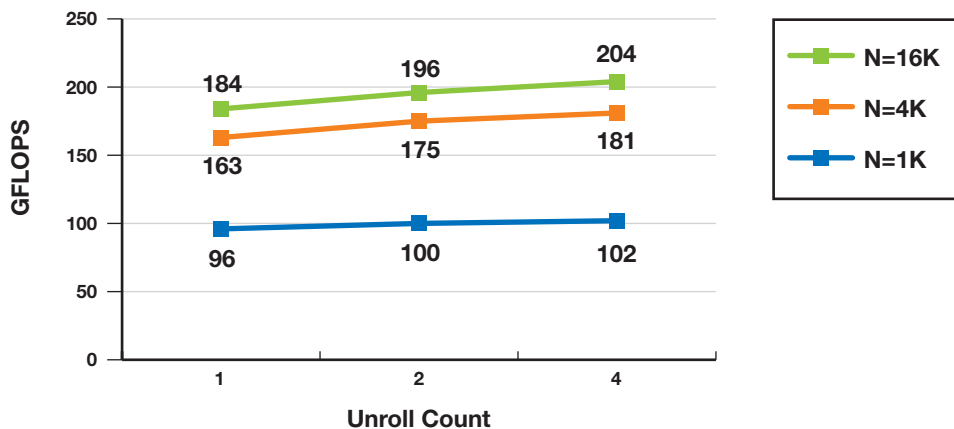


Figure 31-5. Performance Increase with Loop Unrolling

This graph shows the effect of unrolling a loop by replicating the body of the loop 1, 2, and 4 times for simulations with 1024, 4096, and 16,384 bodies. The performance increases, as does register usage, until the level of multiprocessing drops with an unrolling factor of 8.

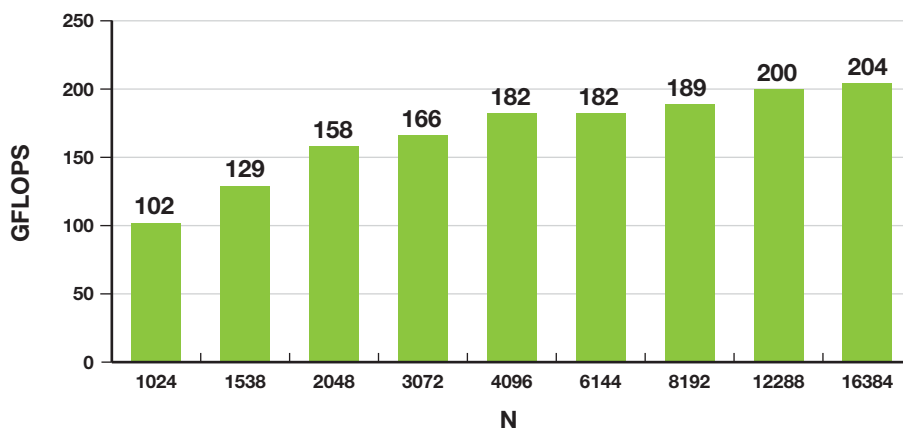


Figure 31-6. Performance Increase as N Grows

This graph shows observed gigaflop rates for varying problem sizes, where each pair-wise force calculation is considered to require 20 floating-point operations. There are evident inefficiencies when $N < 4096$, but performance is consistently high for $N \geq 4096$.

Performance Increase as Block Size Varies

Another performance-tuning parameter is the value of p , the size of the tile. The total memory fetched by the program is N^2/p for each integration time step of the algorithm, so increasing p decreases memory traffic. There are 16 multiprocessors on the

GeForce 8800 GTX GPU, so p cannot be arbitrarily large; it must remain small enough so that N/p is 16 or larger. Otherwise, some multiprocessors will be idle.

Another reason to keep p small is the concurrent assignment of thread blocks to multiprocessors. When a thread block uses only a portion of the resources on a multiprocessor (such as registers, thread slots, and shared memory), multiple thread blocks are placed on each multiprocessor. This technique provides more opportunity for the hardware to hide latencies of pipelined instruction execution and memory fetches. Figure 31-7 shows how the performance changes as p is varied for $N = 1024$, 4096, and 16,384.

Improving Performance for Small N

A final optimization that we implemented—using multiple threads per body—attempts to improve performance for $N < 4096$. As N decreases, there is not enough work with one thread per body to adequately cover all the latencies in the GeForce 8800 GTX GPU, so performance drops rapidly. We therefore increase the number of active threads by using multiple threads on each row of a body's force calculation. If the additional threads are part of the same thread block, then the number of memory requests increases, as does the number of warps, so the latencies begin to be covered again. Our current register use limits the number of threads per block to 256 on the 8800 GTX GPU (blocks of 512 threads fail to run), so we split each row into q segments, keeping $p \times q \leq 256$.

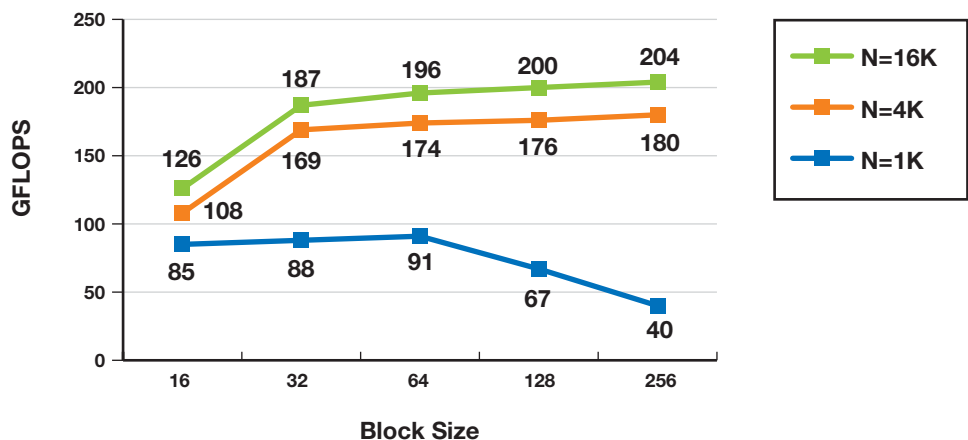


Figure 31-7. Performance as Block Size Varies

This graph shows how performance changes as the tile size p changes, for $N = 1024$, 4096, and 16,384. Larger tiles lead to better performance, as long as all 16 multiprocessors are in use, which explains the decline for $N = 1024$.

Splitting the rows has the expected benefit. Using two threads to divide the work increased performance for $N = 1024$ by 44 percent. The improvement rapidly diminishes when splitting the work further. And of course, for $N > 4096$, splitting had almost no effect, because the code is running at nearly peak performance. Fortunately, splitting did not reduce performance for large N . Figure 31-8 shows a graph demonstrating the performance gains.

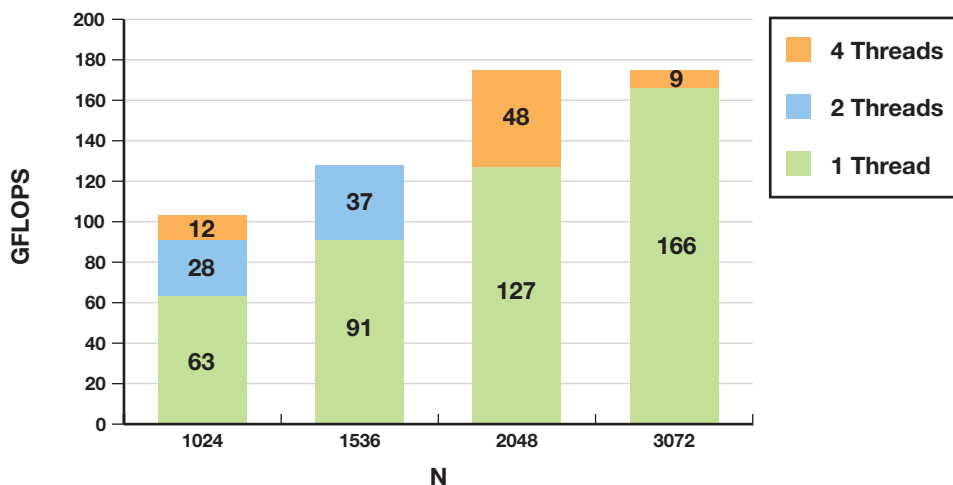


Figure 31-8. Performance Increase by Using Multiple Threads per Body
This chart demonstrates that adding more threads to problems of small N improves performance. When the total force on one body is computed by two threads instead of one, performance increases by as much as 44 percent. Using more than four threads per body provides no additional performance gains.

31.4.2 Analysis of Performance Results

When we judge the performance gain of moving to the GeForce 8800 GTX GPU, the most surprising and satisfying result is the speedup of the N -body algorithm compared to its performance on a CPU. The performance is much larger than the comparison of peak floating-point rates between the GeForce 8800 GTX GPU and Intel processors. We speculate that the main reason for this gain is that Intel processors require dozens of unpipelined clock cycles for the division and square root operations, whereas the GPU has a single instruction that performs an inverse square root. Intel's Streaming SIMD Extensions (SSE) instruction set includes a four-clock-cycle $1/\text{sqrt}(x)$ instruction (vec-

tor and scalar), but the accuracy is limited to 12 bits. In a technical report from Intel (Intel 1999), a method is proposed to increase the accuracy over a limited domain, but the cost is estimated to be 16 clock cycles.

31.5 Previous Methods Using GPUs for N-Body Simulation

The N-body problem has been studied throughout the history of computing. In the 1980s several hierarchical and mesh-style algorithms were introduced, successfully reducing the $O(N^2)$ complexity. The parallelism of the N-body problem has also been studied as long as there have been parallel computers. We limit our review to previous work that pertains to achieving high performance using GPU hardware.

In 2004 we built an N-body application for GPUs by using Cg and OpenGL (Nyland, Harris, and Prins 2004). Although Cg presented a more powerful GPU programming language than had previously been available, we faced several drawbacks to building the application in a graphics environment. All data were either read-only or write-only, so a double-buffering scheme had to be used. All computations were initiated by drawing a rectangle whose pixel values were computed by a shader program, requiring $O(N^2)$ memory. Because of the difficulty of programming complex algorithms in the graphics API, we performed simple brute-force computation of all pair-wise accelerations into a single large texture, followed by a parallel sum reduction to get the vector of total accelerations. This sum reduction was completely bandwidth bound because of the lack of on-chip shared memory. The maximum texture-size limitation of the GPU limited the largest number of bodies we could handle (at once) to 2048. Using an out-of-core method allowed us to surpass that limit.

A group at Stanford University (Elsen et al. 2006) created an N-body solution similar to the one described in this chapter, using the BrookGPU programming language (Buck et al. 2004), gathering performance data from execution on an ATI X1900 XTX GPU. They concluded that loop unrolling significantly improves performance. They also concluded that achieving good performance when $N < 4096$ is difficult and suggest a similar solution to ours, achieving similar improvement. The Stanford University group compares their GPU implementation to a highly tuned CPU implementation (SSE assembly language that achieves 3.8 gigaflops, a performance metric we cannot match) and observe the GPU outperforming the CPU by a factor of 25. They provide code (written in BrookGPU) and analyze what the code and the hardware are doing. The

GPU hardware they used achieves nearly 100 gigaflops. They also remind us that the CPU does half the number of force calculations of the GPU by using the symmetry of $\mathbf{f}_{ij} = -\mathbf{f}_{ji}$.

Since the release of the GeForce 8800 GTX GPU and CUDA, several implementations of N-body applications have appeared. Two that caught our attention are Hamada and Iitaka 2007 and Portegies Zwart et al. 2007. Both implementations mimic the Gravity Pipe (GRAPE) hardware (Makino et al. 2000), suggesting that the GeForce 8800 GTX GPU replace the GRAPE custom hardware. Their N-body method uses a multiple time-step scheme, with integration steps occurring at different times for different bodies, so the comparison with these two methods can only be done by comparing the number of pair-wise force interactions per second. We believe that the performance we have achieved is nearly two times greater than the performance of the cited works.

31.6 Hierarchical N-Body Methods

Many practical N-body applications use a hierarchical approach, recursively dividing the space into subregions until some criterion is met (for example, that the space contains fewer than k bodies). For interactions within a leaf cell, the all-pairs method is used, usually along with one or more layers of neighboring leaf cells. For interactions with subspaces farther away, far-field approximations are used. Popular hierarchical methods are Barnes-Hut (Barnes and Hut 1986) and Greengard's fast multipole method (Greengard 1987, Greengard and Huang 2002).

Both algorithms must choose how to interact with remote leaf cells. The general result is that many body-cell or cell-cell interactions require an all-pairs solution to calculate the forces. The savings in the algorithm comes from the use of a multipole expansion of the potential due to bodies at a distance, rather than from interactions with the individual bodies at a distance.

As an example in 3D, consider a simulation of 2^{18} bodies (256 K), decomposed into a depth-3 octree containing 512 leaf cells with 512 bodies each. The minimum neighborhood of cells one layer deep will contain 27 leaf cells, but probably many more will be used. For each leaf cell, there are at least $27 \times 512 \times 512$ pair-wise force interactions to compute. That yields more than 7 million interactions per leaf cell, which in our implementation would require less than 1 millisecond of computation to solve. The total time required for all 512 leaf cells would be less than a half-second.

Contrast this with our all-pairs implementation³ on an Intel Core 2 Duo⁴ that achieves about 20 million interactions per second. The estimated time for the same calculation is about 90 seconds (don't forget that the CPU calculates only half as many pair-wise interactions). Even the high-performance implementations that compute 100 million interactions per second require 18 seconds. One way to alleviate the load is to deepen the hierarchical decomposition and rely more on the far-field approximations, so that the leaf cells would be populated with fewer particles. Of course, the deeper tree means more work in the far-field segment.

We believe that the savings of moving from the CPU to the GPU will come not only from the increased computational horsepower, but also from the increased size of the leaf cells, making the hierarchical decomposition shallower, saving time in the far-field evaluation as well. In future work we hope to implement the BH or FMM algorithms, to evaluate the savings of more-efficient algorithms.

31.7 Conclusion

It is difficult to imagine a real-world algorithm that is better suited to execution on the G80 architecture than the all-pairs N-body algorithm. In this chapter we have demonstrated three features of the algorithm that help it achieve such high efficiency:

- Straightforward parallelism with sequential memory access patterns
- Data reuse that keeps the arithmetic units busy
- Fully pipelined arithmetic, including complex operations such as inverse square root, that are much faster clock-for-clock on a GeForce 8800 GTX GPU than on a CPU

The result is an algorithm that runs more than 50 times as fast as a highly tuned serial implementation (Elsen et al. 2006) or 250 times faster than our portable C implementation. At this performance level, 3D simulations with large numbers of particles can be run interactively, providing 3D visualizations of gravitational, electrostatic, or other mutual-force systems.

3. Our implementation is single-threaded, does not use any SSE instructions, and is compiled with gcc. Other specialized N-body implementations on Intel processors achieve 100 million interactions a second (Elsen et al. 2006).

4. Intel Core 2 Duo 6300 CPU at 1.87 GHz with 2.00 GB of RAM.

31.8 References

- Aarseth, S. 2003. *Gravitational N-Body Simulations*. Cambridge University Press.
- Barnes, J., and P. Hut. 1986. "A Hierarchical $O(n \log n)$ Force Calculation Algorithm." *Nature* 324.
- Buck, I., T. Foley, D. Horn, J. Sugerman, K. Fatahalian, M. Houston, and P. Hanrahan. 2004. "Brook for GPUs: Stream Computing on Graphics Hardware." In *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2004)* 23(3).
- Darden, T., D. York, and L. Pederson. 1993. "Particle Mesh Ewald: An $N \log(N)$ Method for Ewald Sums in Large Systems." *Journal of Chemical Physics* 98(12), p. 10089.
- Dehnen, Walter. 2001. "Towards Optimal Softening in 3D N-body Codes: I. Minimizing the Force Error." *Monthly Notices of the Royal Astronomical Society* 324, p. 273.
- Dyer, Charles, and Peter Ip. 1993. "Softening in N-Body Simulations of Collisionless Systems." *The Astrophysical Journal* 409, pp. 60–67.
- Elsen, Erich, Mike Houston, V. Vishal, Eric Darve, Pat Hanrahan, and Vijay Pande. 2006. "N-Body Simulation on GPUs." Poster presentation. Supercomputing 06 Conference.
- Greengard, L. 1987. *The Rapid Evaluation of Potential Fields in Particle Systems*. ACM Press.
- Greengard, Leslie F., and Jingfang Huang. 2002. "A New Version of the Fast Multipole Method for Screened Coulomb Interactions in Three Dimensions." *Journal of Computational Physics* 180(2), pp. 642–658.
- Hamada, T., and T. Iitaka. 2007. "The Chamomile Scheme: An Optimized Algorithm for N-body Simulations on Programmable Graphics Processing Units." *ArXiv Astrophysics e-prints*, astro-ph/0703100, March 2007.
- Hockney, R., and J. Eastwood. 1981. *Computer Simulation Using Particles*. McGraw-Hill.
- Intel Corporation. 1999. "Increasing the Accuracy of the Results from the Reciprocal and Reciprocal Square Root Instructions Using the Newton-Raphson Method." Version 2.1. Order Number: 243637-002. Available online at http://cache-www.intel.com/cd/00/00/04/10/41007_nrmethod.pdf.
- Intel Corporation. 2003. *Intel Pentium 4 and Intel Xeon Processor Optimization Reference Manual*. Order Number: 248966-007.

-
- Johnson, Vicki, and Alper Ates. 2005. "NBodyLab Simulation Experiments with GRAPE-6a and MD-GRAPE2 Acceleration." *Astronomical Data Analysis Software and Systems XIV P3-1-6*, ASP Conference Series, Vol. XXX, P. L. Shopbell, M. C. Britton, and R. Ebert, eds. Available online at http://nbodylab.interconnect.com/docs/P3.1.6_revised.pdf.
- Makino, J., T. Fukushige, and M. Koga. 2000. "A 1.349 Tflops Simulation of Black Holes in a Galactic Center on GRAPE-6." In *Proceedings of the 2000 ACM/IEEE Conference on Supercomputing*.
- NVIDIA Corporation. 2007. *NVIDIA CUDA Compute Unified Device Architecture Programming Guide*. Version 0.8.1.
- Nyland, Lars, Mark Harris, and Jan Prins. 2004. "The Rapid Evaluation of Potential Fields Using Programmable Graphics Hardware." Poster presentation at GP², the ACM Workshop on General Purpose Computing on Graphics Hardware.
- Portegies Zwart, S., R. Belleman, and P. Geldof. 2007. "High Performance Direct Gravitational N-body Simulations on Graphics Processing Unit." *ArXiv Astrophysics e-prints*, astro-ph/0702058, Feb. 2007.
- Verlet, J. 1967. "Computer Experiments on Classical Fluids." *Physical Review* 159(1), pp. 98–103.

GPU Gems 3



Edited by Hubert Nguyen
Foreword by Kurt Akeley



- Full color, hard cover, \$69.99
- Experts from industry and universities
- Available for purchase online

For more information, please visit:

<http://developer.nvidia.com/gpugems3>