

# ***Параллельное программирование для высокопроизводительных систем***

Лекция

**Архитектура и программное обеспечение НРС-  
систем факультета ВМК**

.

**Попова Нина Николаевна**

**доцент кафедры СКИ**

**28 октября 2021 г.**

**Архитектура и программное  
обеспечение массивно-  
параллельной вычислительной  
системы  
IBM Blue Gene / P**

*<http://hpc.cs.msu.ru>*

# Дополнительная информация на сайте

hpc@ctmc | Высокопроизводительные вычисления на ВМК МГУ - Mozilla Firefox

File Edit View History Bookmarks Tools Help

hpc@ctmc | Высокопроизводительные ... x +

hpc.cs.msu.su

hpc@ctmc Высокопроизводительные вычисления на ВМК МГУ

Blue Gene/P Общие вопросы Форум Поддержка Ссылки Регистрация

▼ Blue Gene/P

- Новым пользователям
- Быстрый старт
- Подключение
- Файловая система
- ▶ Разработка программ
- ▶ Запуск заданий
- Программное обеспечение
- Официальная документация
- FAQ
- Серия Blue Gene в мире
- Фотографии
- Публикации

▼ Общие вопросы

- Доступ по SSH
- Выбор пароля
- VPN-подключение

○ Форум


○ Поддержка

## Суперкомпьютер IBM Blue Gene/P на факультете ВМК МГУ

С 2008 года на факультете ВМК МГУ имени М. В. Ломоносова работает суперкомпьютер IBM Blue Gene/P, который является одной из первых систем данной серии среди установленных в мире. Архитектура Blue Gene была предложена компанией IBM в рамках проекта по исследованию возможностей достижения новых рубежей в супервычислениях. Более крупные машины данной серии в настоящее время занимают лидирующие позиции в списке пятисот самых мощных компьютеров мира [Top500](#), а машина Blue Gene/P, установленная на ВМК МГУ, в редакции рейтинга от 18 ноября 2008 года оказалась на [128-м месте](#) (в редакции от 16 ноября 2009 года — 348-е место). В списке самых высокопроизводительных компьютеров стран СНГ, опубликованном 22 сентября 2009 года, она находится на [4-й строчке](#).

Система IBM Blue Gene/P принадлежит к новому семейству суперкомпьютеров, обладающих высокой производительностью, масштабируемостью, возможностью обрабатывать данные большего объема, потребляя при этом значительно меньше энергии и занимая меньшую площадь по сравнению с предыдущими системами.

На факультете ВМК МГУ представлена конфигурация, состоящая из двух стоек, содержащих в общей сложности 2048



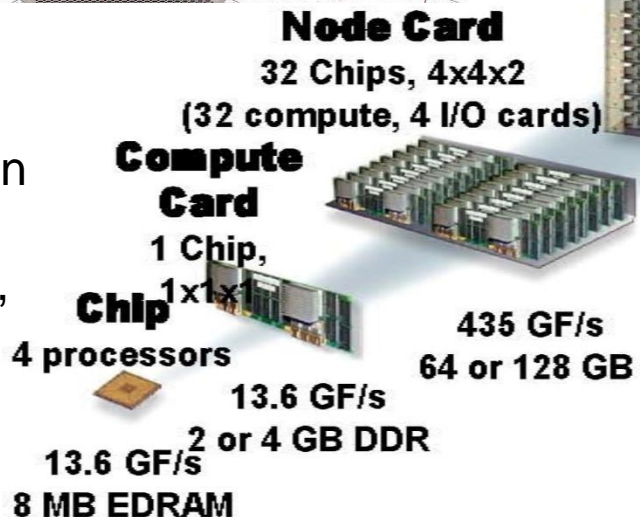
# IBM Blue Gene

- **Blue Gene** –название архитектуры семейства СК
- **Проект начался в 1999 году**
  - IBM, Lawrence Livermore National Laboratory, Department of Energy
  - Первоначальное назначение: расшифровка генома, фолдинг белков
  - Три поколения: BG/L (2001), BG/P (2007), BG/Q (2012)
- **Основные свойства**
  - Масштабируемость (массивно-параллельные системы)
  - Энергоэффективность (1 процессор ~ 40 W лампочка)
- **Реализация**
  - Сравнительно низкая тактовая частота процессора
    - низкое энергопотребление и тепловыделение
    - возможность плотной упаковки в стойку
    - не требует специального кондиционирования
  - Быстрая коммуникационная сеть
    - баланс между скоростью процессора и коммуникации
  - Облегченная ОС (минимизация«шума»)

# Blue Gene/P Hardware



Argonne  
National  
Laboratory, in  
Lemont,  
Illinois, USA,  
2007 .



**Rack**  
32 Node Cards



**Rack**  
Cabled 8x8x16



14 TF/s  
2 or 4 TB

**System**  
Up to 72 Racks



1 PF/s  
144 TB



# Blue Gene P

## 1 стойка

- 1024 четырехъядерных вычислительных узлов
- производительность одного вычислительного узла – 13.6 GF/s
- производительность 1 стойки– 13.6 Tflops
- оперативная память одного узла – 2 GB
- суммарная оперативная память в стойке– 2 TB
- узлов ввода/вывода 8 – 64
- Размеры - 1.22 x 0.96 x 1.96
- занимаемая площадь 1.17 кв.м.
- энергопотребление (1 стойка) - 40 kW (max)

# Конфигурация BlueGene P факультета ВМиК

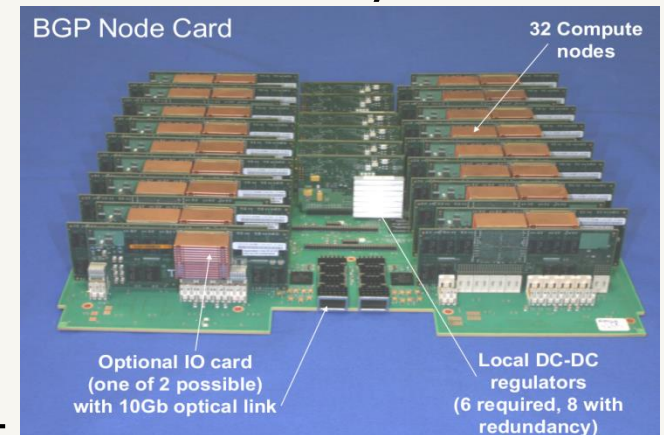
*<http://hpc.cs.msu.ru>*

- 2048 4-ех ядерных узлов
- пиковая производительность 27.2 Tflops
- Реальная производительность по тесту Linpack:
  - 23.2 TFlops
  - 85% от пиковой
- общий объем ОЗУ 4 TB



# Компоненты Blue Gene P

- Основная единица – четырех ядерный вычислительный узел (процессор) , ядро – PowerPC 450 850Mhz + память (2GB)
- Карта узлов (Node card) = 32 вычислительных узла + до 2х узлов ввода-вывода
- Стойка – 32 карты узлов
- Число процессоров в стойке -1024
- Итоговое число ядер на стойку - 4096





# Характеристики вычислительного узла

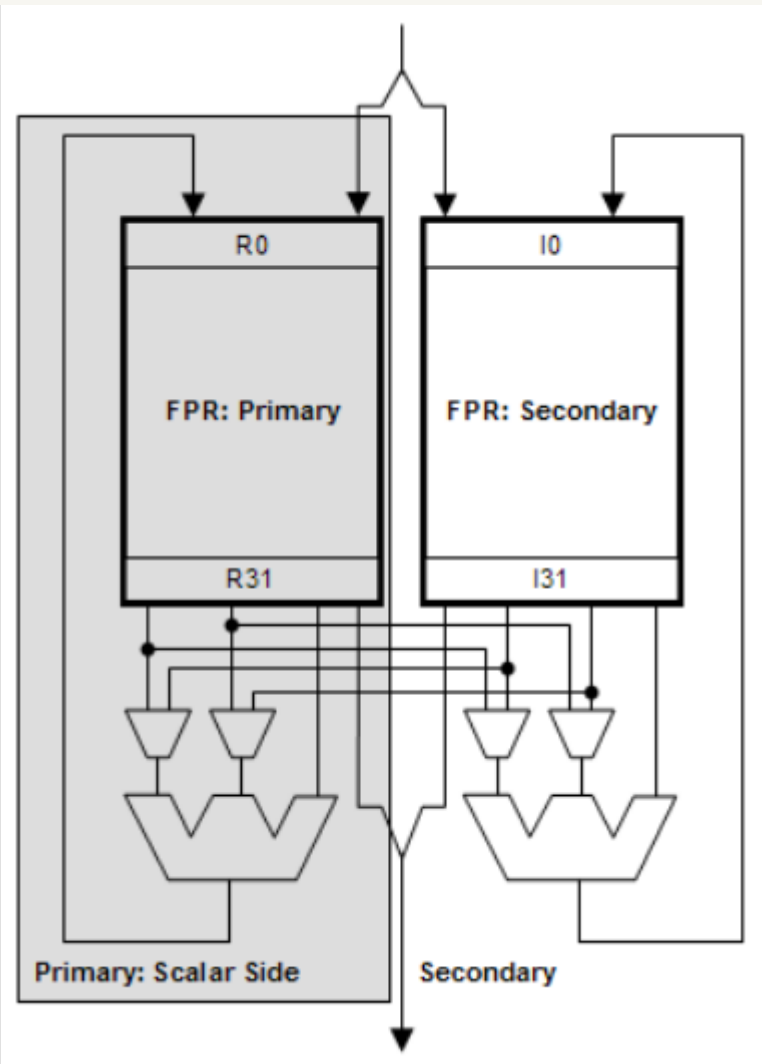
4-ех ядерный 32-битный процессор PowerPC 850 МГц:

- Двойное устройство для работы с вещественными числами с плавающей точкой (double precision)
- 2 Гб памяти
- Работает под управлением облегченной ОС:
  - Создание процессов и управление ими
  - Управление памятью
  - Отладка процессов
  - Ввод-вывод
- **Объем виртуальной памяти равен объему физической**

# Память

- Оперативная память – до 2GB на вычислительный узел, пропускная способность 13.6GBps
- Трёхуровневый кэш:
  - L1 – отдельный для каждого ядра, размер 32Kb
  - L2 – отдельный для каждого ядра, используется для предварительной выборки информации из кэша L1. Считывает\записывает по 16b за одно обращение.
  - L3 – разделен на две части по 4MB, доступ к ним имеют все четыре ядра, для каждого есть канал чтения и канал записи.

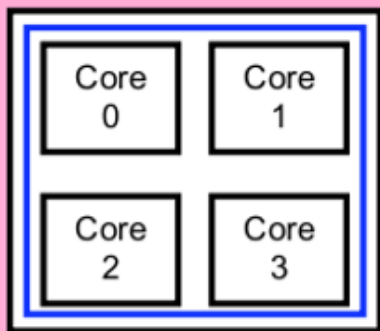
# Double Hammer FPU



- SIMD инструкции могут выполняться одновременно на двух FPU
- Параллельные операции load/store
- Данные **должны** быть выровнены по 16-байтовой границе
  - Иначе производительность будет значительно снижена
  - Даже хуже, чем при использовании только одного FPU
- Компилятор сможет сгенерировать SIMD инструкции, только если данные в памяти расположены подряд (stride-one access)
  - Хотя при более высоких (-O4, -O5) уровнях оптимизации компилятор попытается сгенерировать SIMD инструкции и для данных, расположенных не подряд
  - **-O3 -qarch=450d -qtune=450**

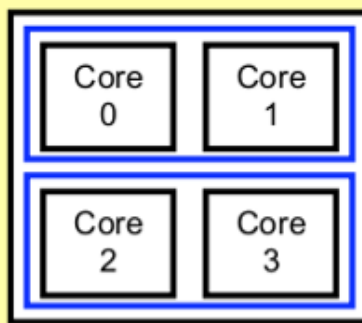
# Режимы выполнения процессов

SMP Mode  
1 Process  
1-4 Threads/Process



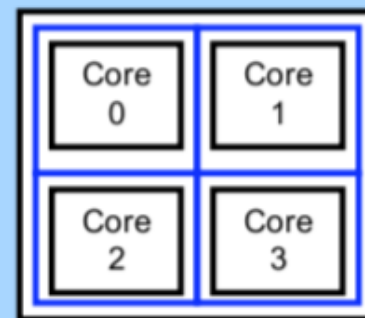
1 MPI процесс , 4 нити,  
2 Гб памяти

Dual Mode  
2 Processes  
1-2 Threads/Process



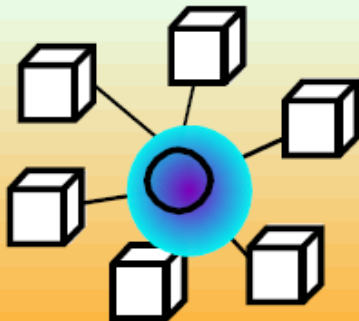
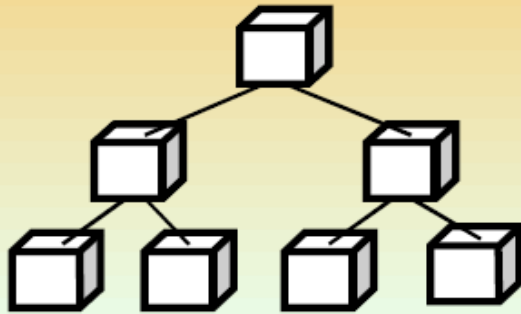
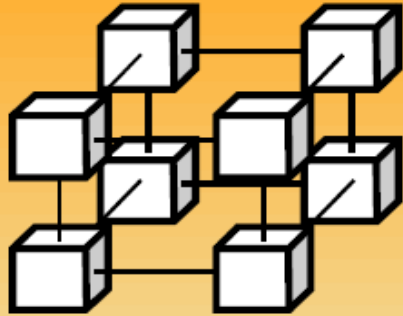
2 MPI процесса , 2  
нити, 1 Гб памяти

Quad Mode (VNM)  
4 Processes  
1 Thread/Process



4 MPI процесс , 1 нить , 512 Кб  
памяти

# Основные коммуникационные сети



## 3-мерный тор

- Виртуальная аппаратная маршрутизация без буферизации
- 3.4 Гбит/с на всех 12 портах (5.1 ГБ/с на узле)
- Задержки между соседними узлами: 0.5 мс (аппаратная), 3 мс (MPI)
- Основная коммуникационная сеть
- Используется в том числе для многих коллективных операций

## Коллективная сеть-дерево

- Для глобальной коммуникации один-ко-всем
- 6.8 ГБ/с на порт
- Задержка на один проход дерева: аппаратная 1.3 мс, MPI 5 мс
- Соединяет все вычислительные узлы и узлы ввода-вывода
- Используется для коллективных операций и коммуникатора MPI\_COMM\_WORLD

## Высокоскоростная сеть для глобальных прерываний

- Задержка на оповещение всех узлов: аппаратная 0.65 мс, MPI 5 мс
- Для MPI\_Barrier

# Компоненты Blue Gene/P

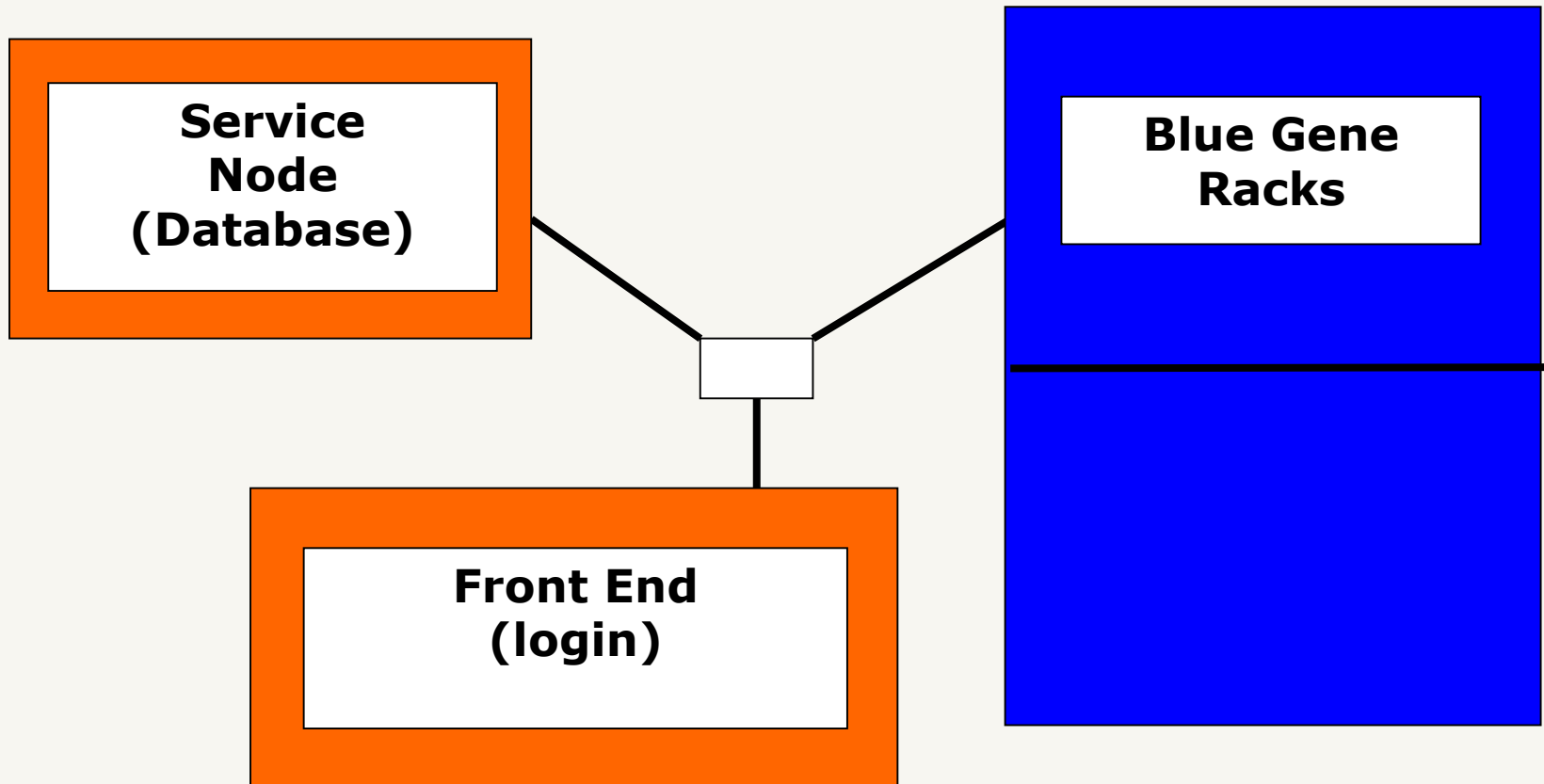
- Помимо вычислительных узлов, в состав системы также входят:
  - узлы ввода-вывода
  - узел управления системой
  - не менее одного узла front end (через них осуществляется доступ пользователей к системе)
  - сеть, связывающая компоненты системы
  - специализированная сеть для сообщения между сервисным узлом и узлами ввода-вывода

# Процессоры ввода-вывода

Отличия по сравнению с вычислительным узлом:

- Установлена полноценная ОС
- Отсутствует подключение к сети тору
- Имеется выход в 10-гигабитную сеть Ethernet

# BlueGene/P





# Состав ПО

- Linux® на узлах ввода\вывода
- MPI (MPICH2) и OpenMP (2.5)
- Стандартное семейство компиляторов IBM XL: XLC/C++, XLF
- Компиляторы GNU
- Система управления заданиями LoadLeveler
- Файловая система GPFS
- Инженерная и научная библиотека подпрограмм (ESSL), математическая библиотека (MASS)

# ОС вычислительного узла

## BlueGene P

- Compute Node Kernel (CNK)
  - “linux-подобная” ОС
  - Нет некоторых системных вызовов (`fork()` в основном). Ограниченная поддержка `mmap()`, `execve()`
  - Минимальное ядро – обработка сигналов, передача системных вызовов к узлам ввода-вывода, старт-завершение задач, поддержка нитей
  - Большинство приложений, которые работают под Linux, портируются на BG/P

# Компиляторы Blue Gene

- IBM XL компиляторы (xlc, xlf77, xlf90)
- работают на front end узлах
  - Fortran: **mpixlf**, **mpixlf90**, **mpixlf95**
  - C: **mpixlc**
  - C++: **mpixlcxx**
- обычно являются скриптами
- GNU компиляторы существуют, но малоэффективны: **mpicc**

# Ключи компилятора XL

- **–qarch=450 –qtune=450**
  - Инструкции только для одного из двух FPU
  - Используйте, если данные не выровнены по 16-байтовой границе
- **–qarch=450d –qtune=450**
  - Выравнивание!
- **–O3 (–qstrict)**
  - Минимальный уровень оптимизации под Double Hammer FPU
- **–O3 –qhot (= –qsimd)**
- **–O4 (–qnoipa)**
- **–O5**
- **–qdebug=diagnostic**
  - Информация по SIMD-инструкциям (только с qhot)
- **–qreport –qlist –qsource**
  - Много полезной информации в .lst файле
- **–qsmp=omp, – qsmp=auto**

# OpenMP

- `_r` суффикс для имени компиляторов например, `mpixlc_r`
- `-qsmp=omp`  
указание компилятору интерпретировать OpenMP директивы
- Автоматическое распараллеливание  
`-qsmp`

# Процессорные партии

- Подмножества вычислительных узлов, выделяемых задаче
- Каждой задаче выделяется своя партия
- Загрузка задачи на исполнение производится независимо от других задач
- Размер партии определяется кратным 32
- (на текущий момент на системе ВМК - кратным 128)
- Для партий размером кратным 512 поддерживается топология тора

## Доступные размеры физических решеток:

- 32: 4 x 4 x 2
- 64: 4 x 4 x 4
- 128: 4 x 4 x 8
- 256: 8 x 4 x 8
- 512: 8 x 8 x 8
- 1024: 8 x 8 x 16
- 1024: 8 x 16 x 8
- 2048: 8 x 16 x 16

Чтобы указать конкретную размерность решетки для 1024 узлов, в командном файле при постановке программы на выполнение необходимо указать параметры

`bg_rotate=false`

`bg_shape=8x8x16` либо `bg_shape=8x16x8`

# Назначение процессов на процессоры (mapping)

Распределение процессов по процессорам по умолчанию:

- **режим SMP**

XYZT, где

<XYZ> - координаты процесса в торе,

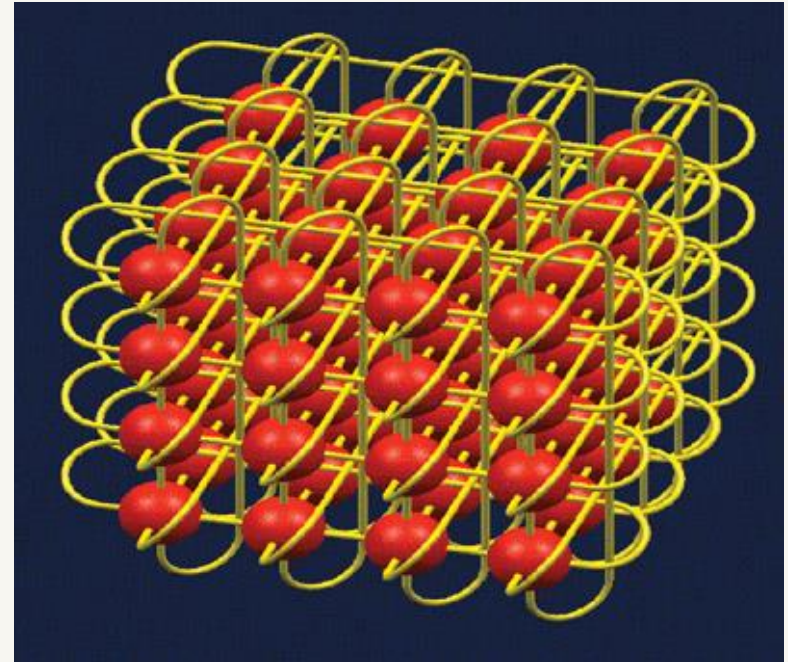
T – номер ядра внутри процесса

Сначала увеличивается X – координата, затем Y и Z-координаты, после этого

**T=0 в режиме SMP**

- **Режим VN и DUAL:**  
TXYZ

Подробная информация о размещении процессов – <http://hpc.cs.msu.ru/bgp/jobs/mapping>





# Mapping

2 способа назначения процессов на процессоры:

- с помощью аргумента командной строки  
***-mapfile TXYZ*** (задаем порядок TXYZ или другие перестановки X,Y,Z,T: TYXZ, TZXY и т.д.)
- указанием map- файла в командной строке  
***-mapfile my.map***, где my.map – имя файла.
- Синтаксис файла распределения – четыре целых числа в каждой строке задают координаты для каждого MPI-процесса (первая строка задает координаты для процесса с номером 0, вторая строка – для процесса с номером 1 и т.д.).

0 0 0 1

0 0 1 1

*Очень важно, чтобы этот файл задавал корректное распределение, с однозначным соответствием между номером процесса и координатами <X, Y, Z, T>.*

# Система управления заданиями

## LoadLeveler

- Оптимизирует использование имеющихся вычислительных ресурсов:
  - Учет приоритета задач и пользователей
  - Динамическое распределение ресурсов
- Управляет очередью заданий
- Для постановки задачи на выполнение требуется указать командный файл (скрипт с расширением .jcf)
- Скрипт для пользователей факультетской системы **mpisubmit.bg:**  
***mpisubmit.bg [SUBMIT\_OPTIONS] EXECUTABLE [-- EXE\_OPTIONS]***
  - n NUMBER — число запрашиваемых узлов
  - m MODE — режим (SMP, DUAL, VN)
  - w HH:MM:SS — время
  - e ENV\_VARS — переменные окружения

***mpisubmit.bg -help***

# Основные команды LL

**llq** - просмотр очереди

**llcancel** – удаление задачи из очереди

**llmap** - просмотр занятости узлов

```
popova@fen1:~/SCMT_2020> mpisubmit.bg -m smp -n 400 -e "OMP_NUM_THREADS=4" ./cannon_omp
llsubmit: Stdin job command file written to "/tmp/loadlx_stdin.25519.ogJFnA".
submit filter in use[ popova fen1.11182684 ]
llsubmit: Processed command file through Submit Filter: "/etc/LoadL/cmc_submit_filter".
llsubmit: The job "fen1.bg.cmc.msu.ru.11182684" has been submitted.
popova@fen1:~/SCMT_2020> ддй
-bash: ддй: command not found
popova@fen1:~/SCMT_2020> llq
```

| Id              | Owner      | Submitted   | ST | PRI | Class     | Running On |
|-----------------|------------|-------------|----|-----|-----------|------------|
| fen1.11182684.0 | popova     | 9/29 16:22  | R  | 50  | n512_m30  | fen1       |
| fen1.11173777.0 | edu-cmc-sk | 12/19 07:39 | I  | 50  | n2048_m03 |            |
| fen1.11179863.0 | edu-cmc-sk | 1/16 23:32  | I  | 50  | n2048_m03 |            |

```
3 job step(s) in queue, 2 waiting, 0 pending, 1 running, 0 held, 0 preempted
```

# Компиляция программ

- **MPI :**

C:            mpixlc -O3 hw.c -o hw

C++:        mpixlcxx -O3 hw.cxx -o hw

Fortran: mpixlf90 -O3 hw.F90 -o hw

- **MPI + OpenMP:**

C:            mpixlc\_r -qsmp=omp -O3 hw.c -o hw

C++:        mpixlcxx\_r -qsmp=omp -O3 hw.cxx -o hw

Fortran: mpixlf90\_r -qsmp=omp -O3 hw.F90 -o hw

# Постановка на счет

MPI + OpenMP, 32 nodes, 15 mins

- SMP (4 потока на MPI процесс):

***mpisubmit.bg -n 32 -m SMP -w 00:15:00 -e "OMP\_NUM\_THREADS=4" hello***

- DUAL (2 потока в MPI процессах):

***mpisubmit.bg -n 32 -m DUAL -w 00:15:00 -e "OMP\_NUM\_THREADS=2" hello***

- VN (1 поток в MPI процессах):

***mpisubmit.bg -n 32 -m VN -w 00:15:00 -e "OMP\_NUM\_THREADS=1" hello***

# Рекомендации по отладке программ

- На Blue Gene нет виртуальной памяти, поэтому необходимо тщательно следить за утечками памяти
  - Application killed with signal 6
    - Недостаточно памяти
- Ключ компилятора -g
- addr2line
  - Утилита для анализа core файлов
- Application killed with signal 7 (memory alignment error)
  - BG\_MAXALIGNEXP=1
    - процесс умирает при первой ошибке выравнивания (полезно при оптимизации)
- –BG\_MAXALIGNEXP=-1
  - игнорировать ошибки выравнивания

# Основной шаблон протокола работы пользователя (1)

## 1. Выход на BGP:

***%ssh <опции> <логин>@ bluegene.hpc.cs.msu.ru***

Например:

***%ssh -i ~/.ssh/bgp.ppk user1@bluegene.hpc.cs.msu.ru***

## 2. Копирование файлов с локального компьютера на Blue Gene/P: (локальная машина)

***%scp -i ~/.ssh/bgp.ppk hw.cpp user1@bluegene.hpc.cs.msu.ru:~/hw.cpp***

# Основной шаблон протокола работы пользователя (2)

3. Компиляция MPI-программы на языке С или С++ : (BGP, front-end)

***%mpixlc hw.c -o hw***

***%mpixlcxx hw.cpp -o hw***

***%mpixlf90 hw.f90 -o hw***

4. Компиляция гибридной MPI-OpenMP программы:

***%mpixlc\_r -qsmp=omp hw.c -o hw***

***% mpixlcxx\_r -qsmp=omp hw.cpp -o hw***



# Основной шаблон протокола работы пользователя (3)

5. Постановка MPI-программы в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **VN** с параметром командной строки :

***%mpisubmit.bg -w 00:15:00 -m VN -n 128 prog - 0.1 200***

6. Постановка MPI+OpenMP программы **prog** в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **SMP** с 4 нитями на каждом узле и с параметром командной строки **parameter**:

***%mpisubmit.bg -w 00:15:00 -m SMP -n 128  
-e «OMP\_NUM\_THREADS=4» prog -- parameter***

# Основной шаблон протокола работы пользователя (4)

7. Постановка MPI+OpenMP программы **prog** в очередь задач с лимитом выполнения 15 минут на 128 узлов в режиме **SMP** с 4 нитями на каждом узле и с параметром командной строки:

```
%mpisubmit.bg -w 00:30:00 -m SMP -n 1024 -mapfile XYZT  
-e «OMP_NUM_THREADS=4» example -- 100
```

# Пример. Матричное умножение.

```
#include <stdio.h>
#include <stdlib.h>
#include <sys/time.h>
#define N 1024
int main(int argc, char **argv) {
    int i, j, k;
    double A[N][N], B[N][N], C[N][N];
    unsigned int iseed = 0;
    struct timeval start;
    struct timeval stop;
    double time;  const double micro = 1.0e-06;
```

# Пример. Матричное умножение.

```
srand(iseed);
for(i=0;i<N;i++)
    for(j=0;j<N;j++) {
        A[i][j]=5-( 10.0 * rand() / ( RAND_MAX + 1.0 ) );
        B[i][j]=5-( 10.0 * rand() / ( RAND_MAX + 1.0 ) );
        C[i][j]=0.0;    }
gettimeofday(&start,NULL);
for (i=0; i<N; i++)
    for (k=0; k<N; k++)
        for(j=0; j<N; j++)
            C[i][j]+=A[i][k]*B[k][j];
```

# Пример. Матричное умножение.

```
gettimeofday(&stop,NULL);

time = (stop.tv_sec - start.tv_sec) + micro*(stop.tv_usec -
start.tv_usec);
printf("Time=%f\n",time);
return 0;
}
```

# Анализ последовательной программы

- Компиляция

*bgxlc\_r -O2 -qtune=450 -qarch=450 mxm.c -o mxm*

- Запуск последовательной программы на одном узле

*mpisubmit.bg -n 1 mxm*

- Результат:

16.799s (~128 MFlop/s, 3.7% от пиковой производительности)

# Последовательная программа.

## Интенсивная оптимизация.

- *bgxlc\_r -O3 -qstrict -qtune=450 -qarch=450 mxm.c -o mxm*

В этом варианте компилятор выполняет некоторые интенсивные оптимизации памяти и времени компиляции в дополнение к тем, которые выполняются с -O2.

Опция -qstrict предоставляется в -O3, чтобы отключить эту агрессивную оптимизацию.

- Результат

6.26s

# Использование Double Hammer инструкций

- *bgxlc\_r -O3 -qstrict -qtune=450 -qarch=450d mxm.c -o mxm*

С такими настройками компилятор попытается сгенерировать SIMD-код для двойного FPU

- 3.496s (~620 MFlop/s, 18% от пиковой производительности)



# Высокоуровневые преобразования

- *bgxlc\_r -O3 -qhot -qtune=450 -qarch=450d -g -qreport -qlist -qsource mxm.c -o mxm*
- В этой настройке компилятор включает такие преобразования, как обмен циклами, разбиение цикла и т. д. Кроме того, для генерации преобразований кода и списков оптимизации использованы параметры компилятора `-qreport` `-qlist` и `-qsource`.
- Файл листинга `mxm.lst` содержит информацию о преобразованном коде. Компилятор распознал схему Matrix Multiply и заменил соответствующие строки кода реализацией DGEMM компилятора XL. Это не всегда желаемое решение, так как результирующая производительность может быть ниже.
- 5.7s

# Использование ESSL DGEMM

- Заменим ijk-цикл на обращение к функции библиотеки ESSL

```
dgemm("T","T",s,s,s,one,&A,s,&B,s,zero,&C,s);
```

- *bgxlc\_r -O3 -qhot -qtune=450 -qarch=450d -g -  
l/opt/ibmmath/essl/4.4/include -c mxm\_dgemm.c*
- *bgxlc\_r -O3 -qhot -qtune=450 -qarch=450d mxm\_dgemm.o -g  
-L/opt/ibmmath/essl/4.4/lib -lesslbg -lesslsmpbg -  
L/opt/ibmcmp/xf/bg/11.1/bglib -lxl90\_r -lxlopt -lxl -lxlmath  
-lxlsmg -o mxm\_dgemm*
- Результат: **0.84s**

# Матричное умножение.

## Параллельный алгоритм Кэннона.

- *mpixlc\_r -O3 -qstrict -qtune=450 -qarch=450d -g cannon.c -o cannon*
- *mpisubmit.bg -n 1024 ./cannon*

Результаты:

- Размер матриц: **16384x16384**
- Количество узлов: 1024
- Время = 23.2 сек.

# Матричное умножение. Кэннона+dgemm

- `mpixlc_r -O3 -qstrict -qtune=450 -qarch=450d -g -  
I/opt/ibmmath/essl/4.4/include -c cannon_dgemm.c`
- `mpixlc_r -O3 -qstrict -qtune=450 -qarch=450d  
cannon_dgemm.o -g -L/opt/ibmmath/essl/4.4/lib -lesslb -  
L/opt/ibmcmp/xlf/bg/11.1/bglib -lxlf90_r -lxlopt -lxl -lxlfmath  
-xlsm -o cannon_dgemm`
- `mpisubmit.bg -n 1024 ./cannon`

Результаты:

- Размер матриц: 16384x16384
- Количество узлов: 1024
- Время = **1.79s** ( было 23.2 сек.)

# Матричное умножение.

## Алгоритм Кэннона, MPI+OpenMP

- `mpixlc_r -O3 -qsmp=omp -qstrict -qtune=450 -qarch=450d -g cannon_omp.c -o cannon_omp`
- `mpisubmit.bg -n 256 -m SMP -e «OMP_NUM_THREADS=2» ./cannon_omp`

Результаты:

- Размер матриц: 16384x16384
- Количество узлов: 256
- Время = **180.8 s**

`mpisubmit.bg -n 256 -m SMP -e «OMP_NUM_THREADS=4» ./cannon_omp`

- Время = **90 s**

`mpisubmit.bg -n 256 -m vn -e «OMP_NUM_THREADS=1» ./cannon_omp`

Время = **89 s**

# Ссылки

- <http://hpc.cs.msu.ru/bgp>
- IBM System Blue Gene Solution: Blue Gene/P Application Development, SG24-7287-00  
[redbooks.ibm.com/abstracts/sg247287.html](http://redbooks.ibm.com/abstracts/sg247287.html)

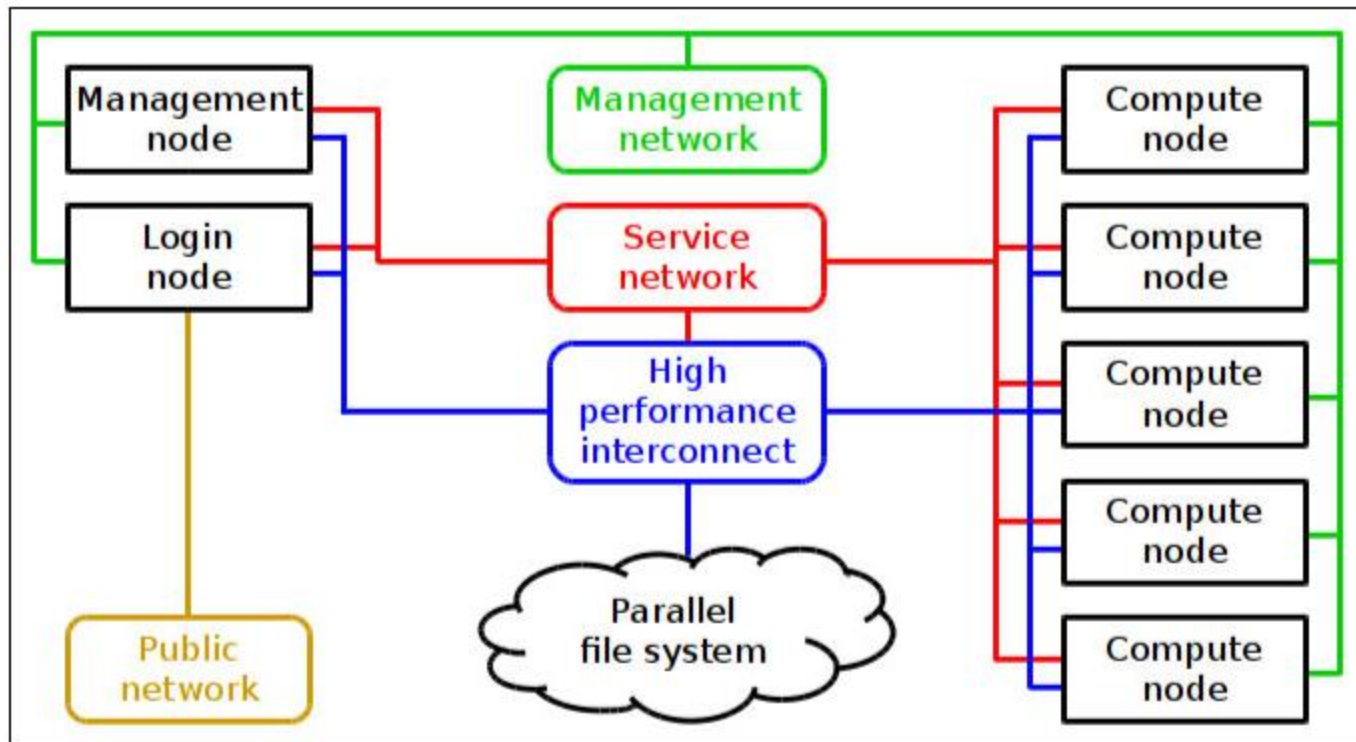


Архитектура и программное  
обеспечение высокопроизводительной  
вычислительной системы

Polus

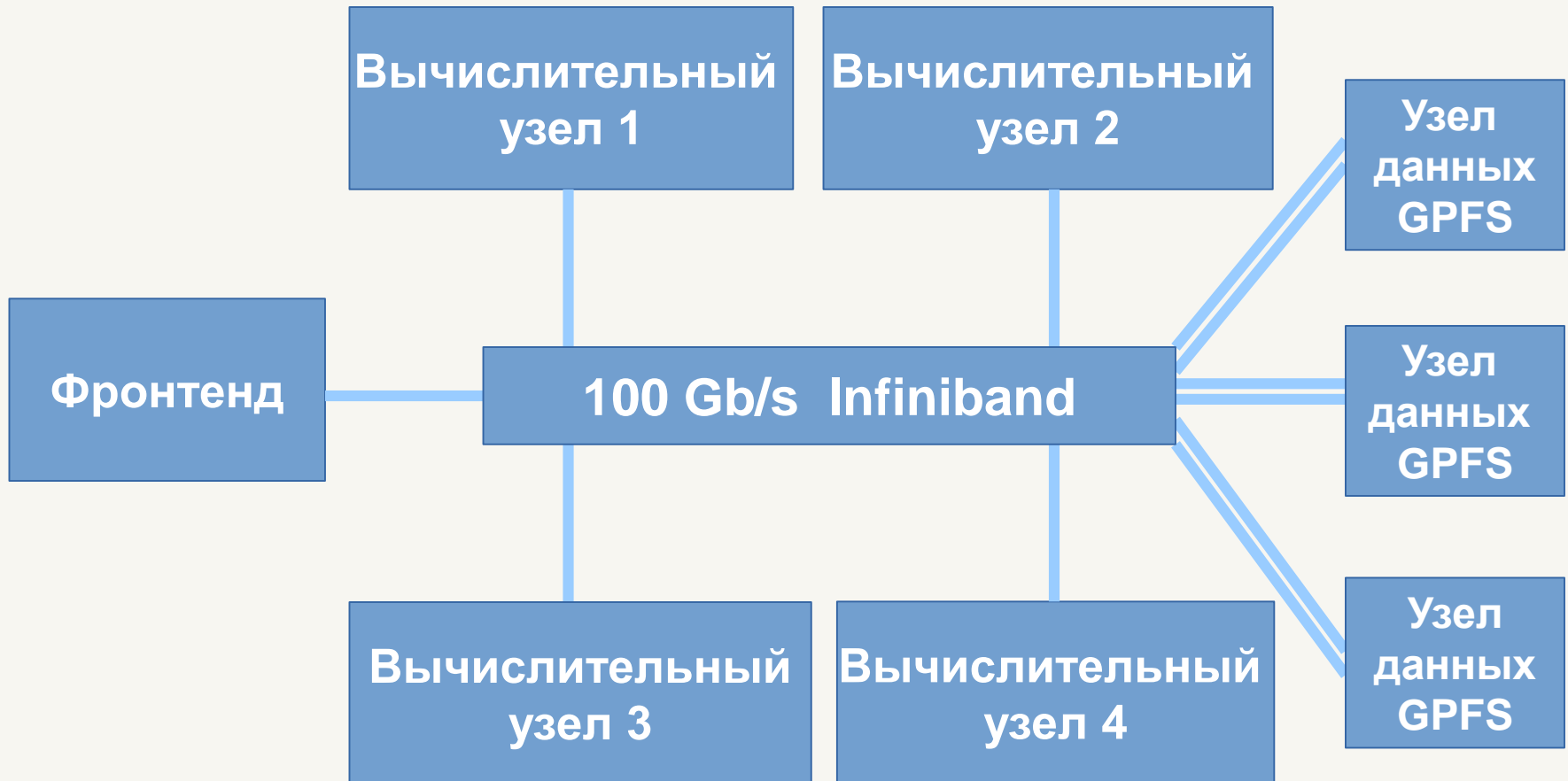
*<http://hpc.cs.msu.ru>*

# Общая схема организации НРС





# Схема ПВС IBM Polus



# Кластер Polus

|                              |                             |
|------------------------------|-----------------------------|
| Пиковая производительность   | 55.84 TFlop/s               |
| Производительность (Linpack) | 40.39 TFlop/s (72% от пика) |
| Вычислительных узлов         | 5                           |

На каждом узле:

|                         |                               |
|-------------------------|-------------------------------|
| Процессоры IBM Power 8  | 2                             |
| NVIDIA Tesla P100       | 2                             |
| Число процессорных ядер | 20                            |
| Число потоков на ядро   | 8                             |
| Оперативная память      | 256 Гбайт (1024 Гбайт узел 5) |
| Коммуникационная сеть   | Infiniband / 100 Gb           |
| Система хранения данных | GPFS                          |
| Операционная система    | Linux Red Hat 7.5             |

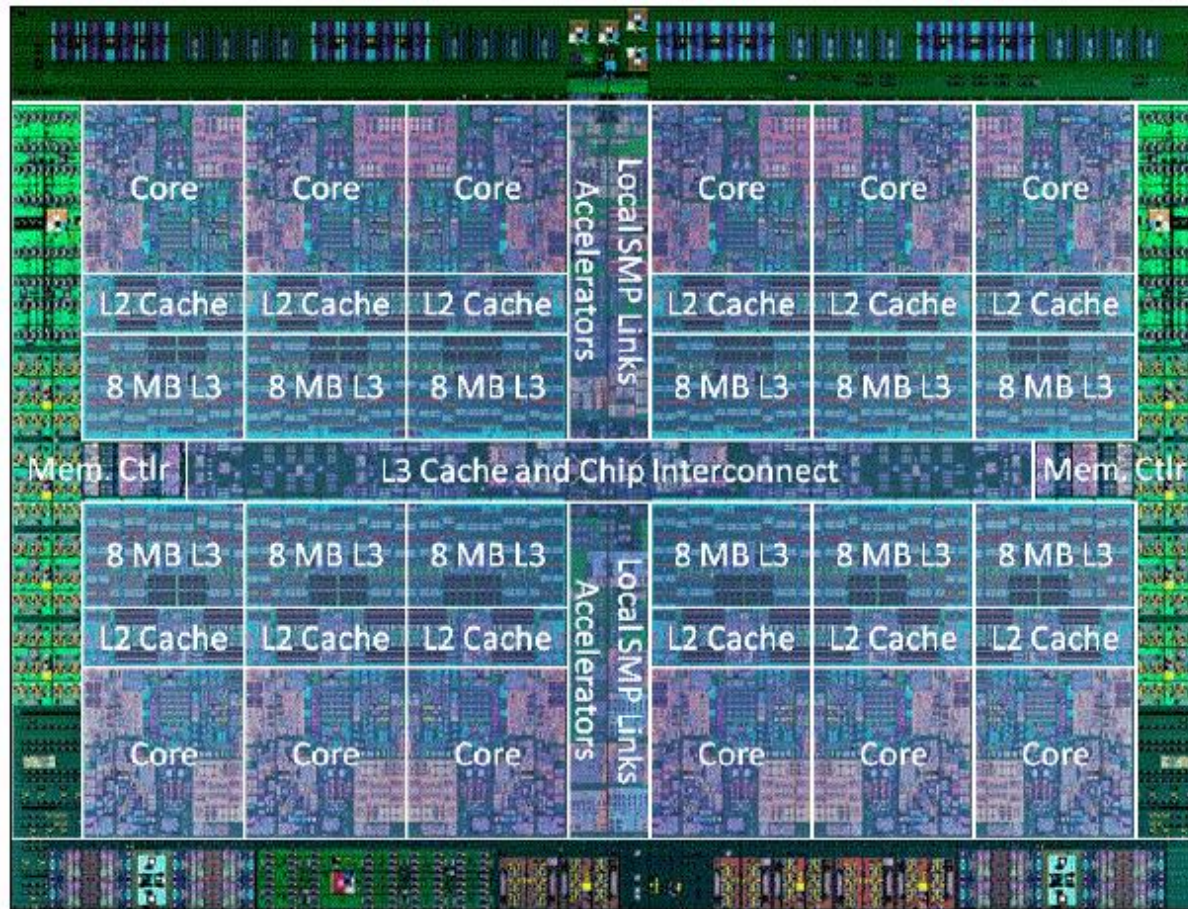
# Характеристика узлов Polus

- Каждый узел имеет 2 сокета с 10 ядрами на сокет = 20 ядер на узел.
- Узлы используются в SMT8 режиме => 8 аппаратных потоков на ядро => 160 потоков на узел.
- Linux трактует каждый поток как логический CPU.
- Размещение и привязка потоков важны для хорошей производительности.
- Существенное влияние NUMA природы на доступ к памяти: лучше всего память, являющаяся локальной к сокету.

# Организация кеш-памяти в Power8

- 64 KB L1 кеш данных на ядро, 3-5 тактов время доступа
- 32 KB L1 кеш инструкций на ядро.
- 512 KB L2 кеш на ядро, ~12 тактов время доступа.
- 8 MB L3 кеш на ядро (NUMA архитектура), ~27 тактов.

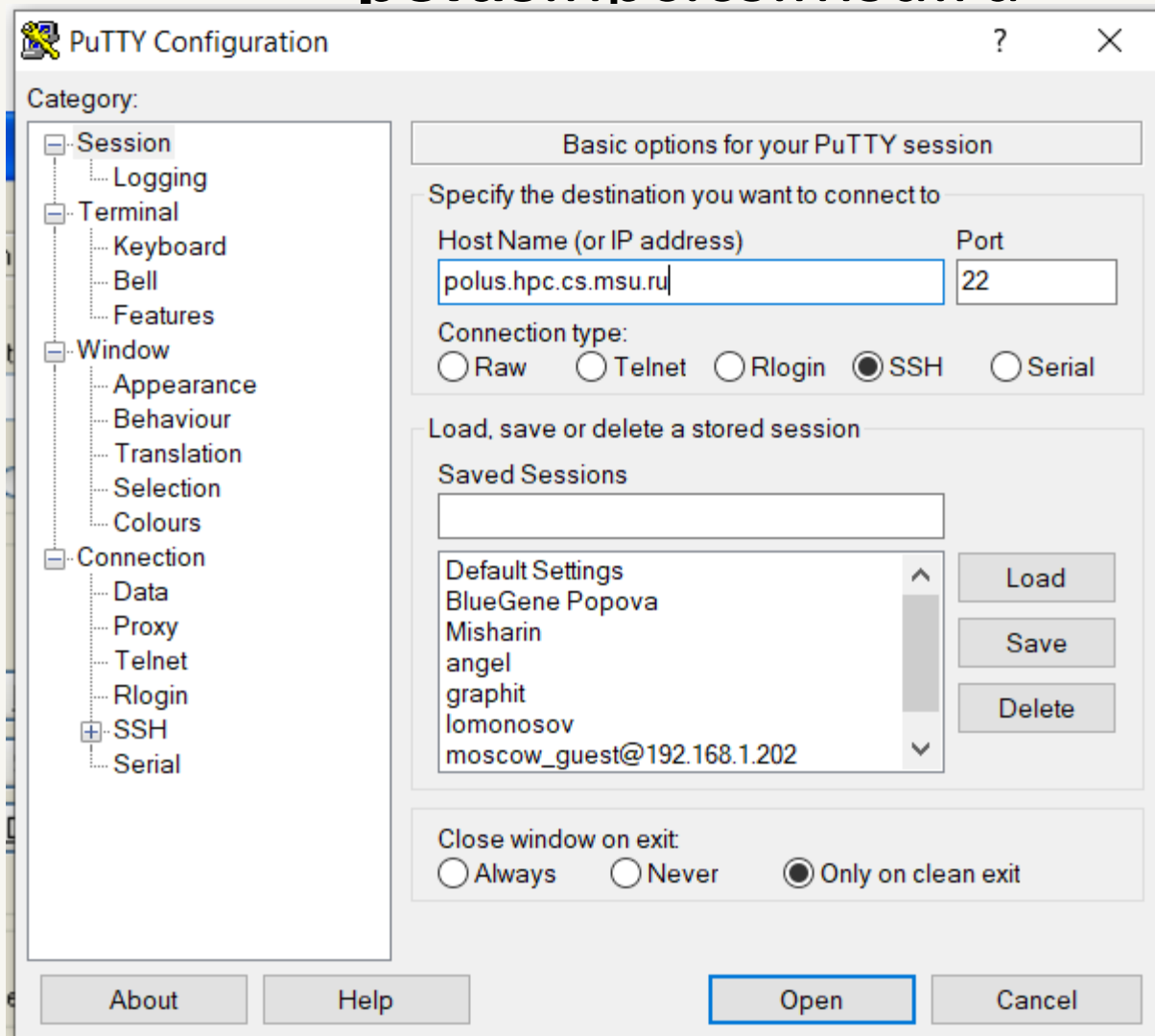
# Схема процессора Power8



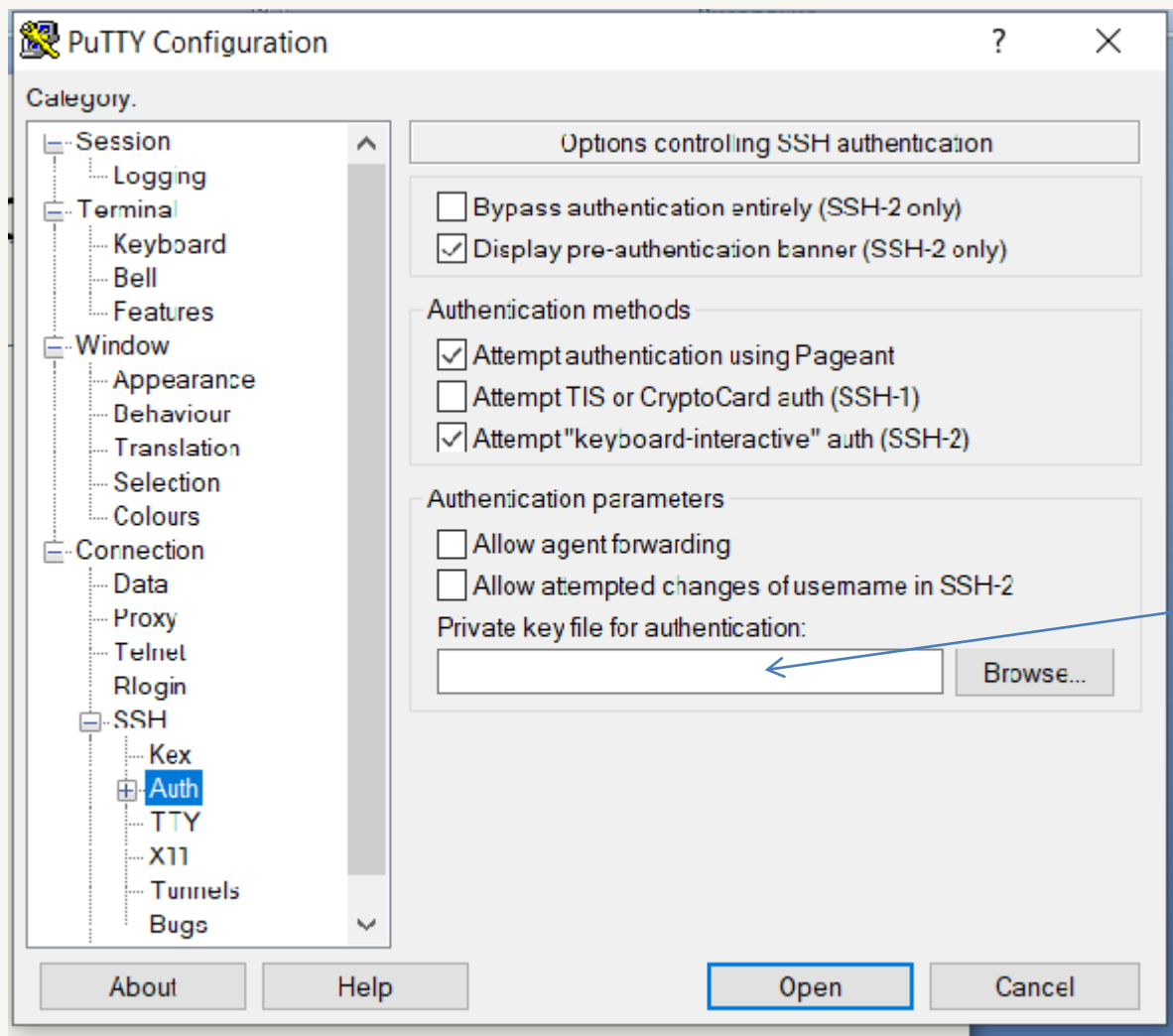
# Организация работы пользователя на ВС Polus

# Доступ на Polus. Putty

## polus.hpc.cs.msu.ru



# Доступ на Polus



Здесь указываете  
файл с приватным  
ключом



# Инфраструктура modules

\$ *module list* (показывает, какие модули загружены)  
\$ *module avail* (доступные модули)  
\$ *module add module\_name* (добавляет нужный модуль)  
\$ *module unload module\_name* (удаляет модуль)

Для создания и запуска MPI программ надо загрузить модуль SpectrumMPI.

```
% module load SpectrumMPI
```

Можно использовать OpenMPI

```
% module load OpenMPI
```

# Компиляторы

- **Компиляторы** : clang, gnu, ibm xl ... nvcc  
gcc, g++, gfortran : по умолчанию 4.8.5;
- Некоторые полезные опции  
-mcpu=power8 , -Ofast, -fopenmp
- Компиляторы GNU вызываются по умолчанию для nvcc
- Компиляция OpenMP-программы:  
*%xlc\_r -qsmp=smp <другие опции> -o <executable name>*
- Компиляция гибридной MPI+OpenMP-программы:  
*%mpixlc\_r -qsmp=smp <другие опции> -o <executable name>*

# Постановка заданий на счет

- Скрипт **mpisubmit.pl**

*%mpisubmit.pl [параметры скрипта] исполняемый\_файл  
[-- параметры исполняемого файла]*

- Параметры скрипта:

*-p <число процессов>*

*-t <число нитей>*

*-stdin <имя файла>*

*-h* - выдает список опций

# Постановка OpenMP-заданий на счет

*%mpisubmit.pl -p 1 -t 8 <исполняемый\_файл>*

*[-- параметры исполняемого файла]*

- Если требуемое число нитей больше 8, параметр *p* должен быть кратным 4.
- Если на одном узле надо использовать больше 8 нитей – использовать скрипт *OpenMP\_job.lsf*

```
#BSUB -n M  
#BSUB -W 00:15  
#BSUB -o my_job.%J.out  
#BSUB -e my_job.%J.err  
#BSUB -R "span[hosts=1]"  
OMP_NUM_THREADS=N ./my_job
```

$N \leq 160$  – число потоков  
 $M = [N/8] + 1$  – число ядер

*%bsub < OpenMP\_job.lsf*

# Система управления заданиями LSF.

## Основные команды

- ***bsub <job>*** : постановка задания в очередь
- ***bjobs*** : вывод списка заданий пользователя (по умолчанию формат вывода задается установкой LSB\_BJOBS\_FORMAT)
- ***bjobs uall*** : список всех задач
- ***bjobs I <jobid>*** : вывод информации по конкретной задаче jobid
- ***bkill <jobid>*** : запрос к LSF на удаление задачи

# Система управления заданиями LSF.

## Командные скрипты.

Пример: Командный скрипт для постановки OpenMP-программ в очередь на выполнение

```
#An example of LSF file. OpenMP_job.lsf  
#BSUB -J "OpenMP_job"  
#BSUB -o "OpenMP_job%J.out"  
#BSUB -e "OpenMP_job%J.err"  
#BSUB -R "affinity[core(N)]"  
/polusfs/lsf/openmp/launchOpenMP.py ./Job_OMP
```

← Комментарий

← Специальный скрипт для привязки потока к ядру

Использование:

**%bsub <OpenMP\_job.lsf**

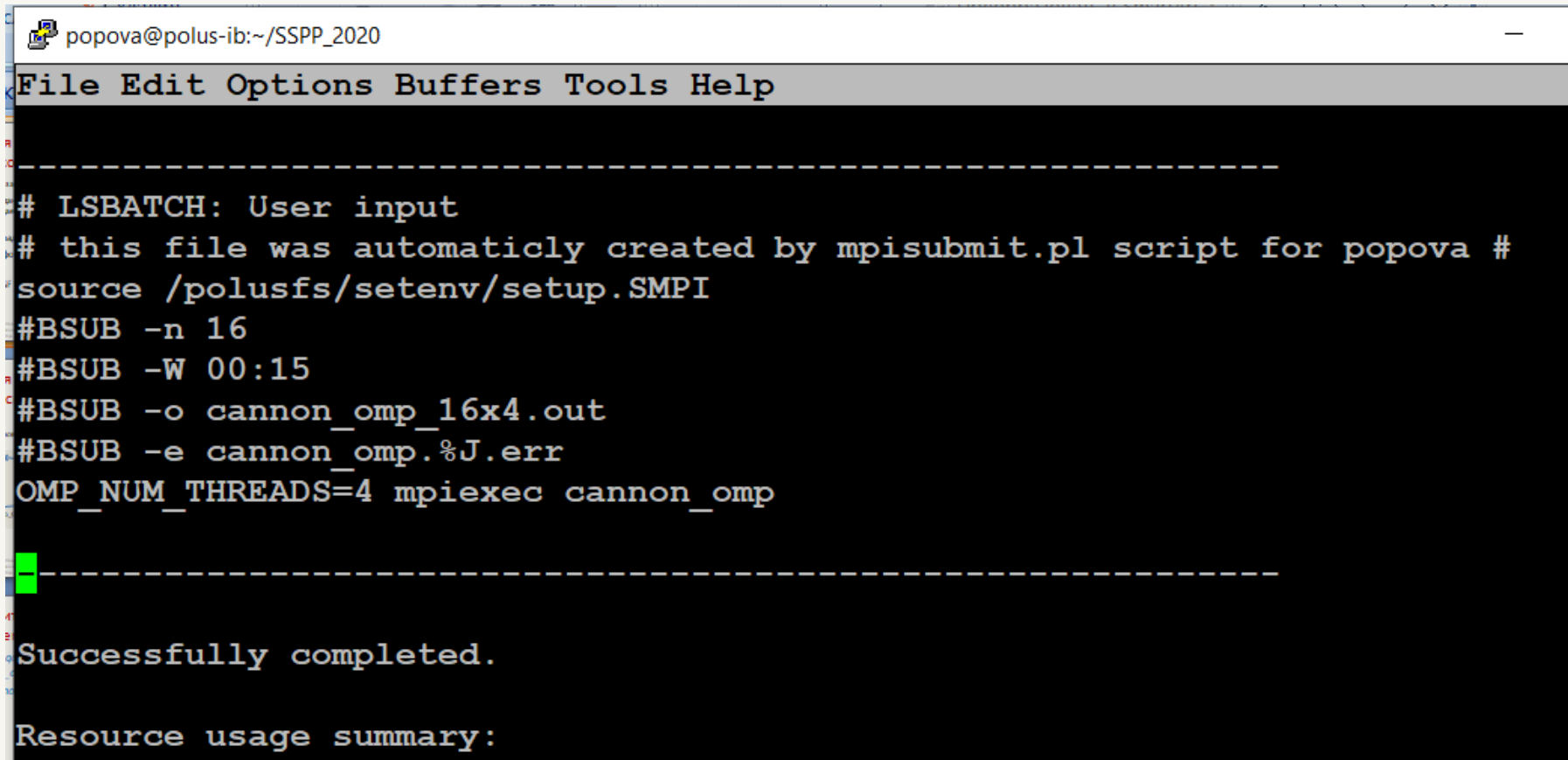
# Пример. Алгоритм Кэннона, MPI+OpenMP

- `mpixlc -O3 -qsmpr=omp -qarch=pwr8 cannon_omp.c -o cannon_omp`
- `mpisubmit.pl -p 16 -t 4 --stdout cannon_omp_16x4.out cannon_omp`

```
[popova@polus-ib SSPP_2020]$ mpisubmit.pl -p 16 -t 4 --stdout cannon_omp_16x4.out cannon_omp
Job <639344> is submitted to default queue <short>.
[popova@polus-ib SSPP_2020]$ bjobs
```

| JOBID  | USER         | STAT                 | QUEUE     | SLOTS       | NALLOC | SLOT | JOB_NAME                       |
|--------|--------------|----------------------|-----------|-------------|--------|------|--------------------------------|
|        | SUBMIT_TIME  | ESTIMATED_START_TIME | TIME_LEFT | PEND_REASON |        |      |                                |
| 639344 | popova       | RUN                  | short     | 16          | 16     |      | *_THREADS=4 mpiexec cannon omp |
|        | Sep 29 22:36 |                      | -         | 0:11        | L      | -    |                                |

# Polus. Результат выполнения.



```
popova@polus-ib:~/SSPP_2020
File Edit Options Buffers Tools Help
-----
# LSBATCH: User input
# this file was automaticly created by mpisubmit.pl script for popova #
source /polusfs/setenv/setup.SMPI
#BSUB -n 16
#BSUB -W 00:15
#BSUB -o cannon_omp_16x4.out
#BSUB -e cannon_omp.%J.err
OMP_NUM_THREADS=4 mpiexec cannon_omp
-----
Successfully completed.

Resource usage summary:
```



# Polus. Результат выполнения.

```
CPU time : 3467.97 sec.  
Max Memory : 9062 MB  
Average Memory : 8430.56 MB  
Total Requested Memory : -  
Delta Memory : -  
Max Swap : -  
Max Processes : 20  
Max Threads : 105  
Run time : 439 sec.  
Turnaround time : 433 sec.
```

The output (if any) follows:

```
Number of MPI processes =16  
Number of OMP threads =1  
Time: 429.9715s
```