Московский государственный университет имени
М.В.Ломоносова
Факультет вычислительной математики и кибернетики

Отчёт

Отказоустойчивость MPI-программы

Зыкин Ярослав
428 группа

10 января 2021 г.

Содержание

1 Постановка задачи

Доработать MPI-программу, реализованную в рамках курса "Суперкомпьютеры и параллельная обработка данных":

1. Добавить контрольные точки для продолжения работы программы в случае сбоя.

2. Реализовать один из 3-х сценариев работы после сбоя:

   (a) продолжить работу программы только на "исправных"процессах;

   (b) вместо процессов, вышедших из строя, создать новые процессы MPI, которые необходимо использовать для продолжения расчетов;

   (c) при запуске программы на счет сразу запустить некоторое дополнительное количество MPI-процессов, которые использовать в случае сбоя.

В этом решении будет использоваться не реализованная в рамках курса "Суперкомпьютеры и параллельная обработка данных"программа, а один из тестов NASA, взятый с сайта. В качестве сценария работы после сбоя будет использоваться третья стратегия: сразу запускать большее количество процессов, которые будут использованы в случае сбоя.

2 Описание используемой программы

NAS Parallel Benchmarks (NPB) - это небольшой набор программ, предназначенных для оценки производительности параллельных суперкомпьютеров. Тесты получены из приложений вычислительной гидродинамики (CFD) и состоят из пяти ядер и трех псевдоприложений в исходной спецификации «pencil-and-paper» (NPB 1). Набор тестов был расширен за счет включения новых тестов для неструктурированных адаптивных сеток, параллельного ввода-вывода, многозонных приложений и вычислительных сеток. Размеры проблем в NPB предопределены и обозначены как разные классы. Эталонные реализации NPB доступны в широко используемых моделях программирования, таких как MPI и OpenMP (NPB 2 и NPB 3).

В качестве дорабатываемой программы будет использован одно из "ядер"NPB – тест IS.

3 Описание решения задачи

В качестве стратегии продолжения работы программы после сбоя было выбрано использование изначально выделенных лишних процессов вместо вышедших из строя. Для реализации этого сценария используется расширение $MPI - ULFM$. При доработке программы было сделано следующее:

1. Добавлен обработчик ошибок MPI – $err\_handler$.
2. Тест IS при подсчётах использует количество процессов, равное степени двойки, остальные процессы помечаются как неиспользуемые и завершаются. Мы же оставшиеся процессы будем использовать как резервные.
3. Для отделения рабочих процессов от резервных и минимизации изменений исходного кода программы, процессы из исходного коммуникатора $comm\_main$ разделяются на две группы, информация о каждой из них помещается в коммуникатор $comm\_work$.
4. Для каждого процесса формируется имя файла, в который будут помещаться контрольные точки.
5. Исходный тест содержит 10 итераций, между ними и будем сохранять контрольные точки. Для экономии времени восстановление будет производиться только в случае ошибки (об этом сигнализирует ненулевое значеие переменной $error\_occured$).
6. Сохраняемая в контрольных точках информация состоит из количества успешно прошедших проверок $passed\_verification$ (бесполезно для счёта, но необходимо для того, чтобы итоговая проверка выполнялась успешно) и рабочего массива key_array.
7. Добавленный обработчик ошибок содержит в себе следующую функциональность:
   (a) Устанавливает ненулевое значение переменной $error\_occured$;
   (b) Обновляет основной коммуникатор $comm\_main$, удаляя из него вышедший из строя процесс с помощью функции ULFM, отсутствующей в стандарте MPI, $MPIX\_Comm\_shrink()$;
   (c) Производит перераспределение процессов по группам, согласно пункту 3;
   (d) Если рабочих процессов остаётся меньше, чем необходимо для продолжении работы, производится $MPI\_Abort()$.
8. Для того, чтобы все процессы находились на одной итерации, добавлены вызовы $MPI\_Barrier()$.
9. После завершения основного цикла все неиспользованные резервные процессы уничтожаются.

10. Моделирование сбоя производится в виде посылки выбранными процессами (в общем случае, произвольного количества, в данном случае – не более двух) после выбранных итераций сигнала $SIGKILL$ самим себе.

4 Запуск программы и исходный код

Написание и тестирование программы с использованием расширения ULFM производилось с помощью Docker. При запуске именно этим способом следует выполнить следующие действия:

1. Убедиться в том, что Docker установлен.
2. Скачать образ ULFM с помощью команды

   ```
   docker pull abouteiller/mpi-ft-ulfm
   ```

3. Перед сборкой и запуском программы выполнить следующие команды:

   ```
   alias make='docker run -v
       $PWD:/sandbox:Z abouteiller/mpi-ft-ulfm make'
   alias mpirun='docker run -v
       $PWD:/sandbox:Z abouteiller/mpi-ft-ulfm mpirun'
   ```

4. Собрать и запустить программу ($C – используемый датасет (S, W, A, B, C, D), $N – количество MPI-процессов):

   ```
   make run CLASS=$C N=$N
   ```

В выводе программы могут содержаться сомнительные строчки вида

```
[...] Read -1, expected ..., errno = 1
```

Однако, согласно официальному гиту openMPI, они появляются из-за взаимодействия систем MPI и Docker и являются безвредными сообщениями.

Основная функциональность добавлена в файл is.c в 329-406 строках.