Нижегородский Государственный Университет им. Н.И. Лобачевского

Общий курс

Теория и практика параллельных вычислений

Лекция 16

Методы разработки параллельных программ для многопроцессорных систем с общей памятью (стандарт OpenMP) – 2

Гергель В.П.

Содержание

Синхронизация…

Директива **master** определяет фрагмент кода, который должен быть выполнен только основным потоком; все остальные потоки пропускают данный фрагмент кода (завершение директивы по умолчанию не синхронизируется)

```
#pragma omp master newline
  structured_block
```

Синхронизация…

Директива **critical** определяет фрагмент кода, который должен выполняться только одним потоком в каждый текущий момент времени (*критическая секция*) основным потоком; все остальные потоки пропускают данный фрагмент кода (завершение директивы по умолчанию не синхронизируется)

```
#pragma omp critical [ name ] newline
  structured_block
```

Синхронизация…

Директива **critical** (пример)

```
#include <omp.h>
main() {
 int x;
 x = 0;
 #pragma omp parallel shared(x)
 {
  #pragma omp critical
    x = x + 1;
 } /* end of parallel section */
}
```

Синхронизация…

Директива **barrier** – определяет точку синхронизации, которую должны достигнуть все процессы для продолжения вычислений (директива должны быть вложена в блок)

```
#pragma omp barrier newline
```

Синхронизация…

Директива **atomic** – определяет переменную, доступ к которой (чтение/запись) должна быть выполнена как неделимая операция

```
#pragma omp atomic newline
   statement_expression
```

- Возможный формат записи выражения
  x binop = expr , x++, ++x, x--, --x
- ***x*** должна быть скалярной переменной
- ***expr*** не должно ссылаться на ***x***
- ***binop*** должна быть неперегруженной операцией вида
  +, -, *, /, &, ^, |, >>, <<

Синхронизация…

Директива **flush** – определяет точку синхронизации, в которой системой должно быть обеспечено единое для всех процессов состояние памяти (т.е. если потоком какое-либо значение извлекалось из памяти для модификации, измененное значение обязательно должно быть записано в общую память)

`#pragma omp flush (list)` ***`newline`***

- Если указан список list, то восстанавливаются только указанные переменные
- Директива **flush** неявным образом присутствует в директивах barrier, critical, ordered, parallel, for, sections, single

Синхронизация…

Директива **ordered** – указывает фрагмент кода параллельного цикла, который должен выполняться точно в таком же порядке, как и при последовательном выполнении

```
#pragma omp ordered newline
  structured_block
```

- В каждый момент времени в блоке **ordered** может находиться только один поток
- На одной итерации цикла может быть только одна директива **ordered** и эта директива может выполниться только однократно
- Цикл, в котором имеется директива **ordered**, должен иметь параметр **ordered**

Синхронизация

Директива **threadprivate** – используется для создания поточных копий для глобальных переменных программы; созданные копии не видимы между потоками, но существуют во все время выполнения программы

```
#pragma omp threadprivate (list)
```

Управление областью видимости данных…

Общие (разделяемые между потоками, *shared*) переменные

– static, переменные с областью видимости в пределах файла

Локальные (*private*) данные потоков

- переменные циклы

Управление областью видимости обеспечивается при помощи параметров (*clause*) директив

private, firstprivate, lastprivate, shared, default, reduction, copyin

которые определяют, какие соотношения существуют между переменными последовательных и параллельных фрагментов выполняемой программы

Управление областью видимости данных…

Параметр **shared** определяет список переменных, которые будут общими для всех потоков параллельной области; правильность использования таких переменных должна обеспечиваться программистом

shared (list)

Параметр **private** определяет список переменных, которые будут локальными для каждого потока; переменные создаются в момент формирования потоков параллельной области; начальное значение переменных является неопределенным

private (list)

Управление областью видимости данных…

Параметр **firstprivate** позволяет создать локальные переменные потоков, которые перед использованием инициализируются значениями исходных переменных
firstprivate (list)

Параметр **lastprivate** позволяет создать локальные переменные потоков, значения которых запоминаются в исходных переменных после завершения параллельной области (используются значения потока, выполнившего последнюю итерацию цикла или последнюю секцию)
lastprivate (list)

Управление областью видимости данных…

Параметр **copyin** позволяет выполнить инициализацию переменных директивы threadprivate

copyin (list)

Параметр **default** устанавливает область видимости переменных по умолчанию

default (shared | none)

Управление областью видимости данных…

Параметр **reduction** определяет список переменных, для которых выполняется операция редукции; перед выполнением параллельной области для каждого потока создаются копии этих переменных, потоки формируют значения в своих локальных переменных и при завершении параллельной области на всеми локальными значениями выполняются необходимые операции редукции, результаты которых запоминаются в исходных (глобальных) переменных

reduction (operator: list)

Управление областью видимости данных…

Параметр **reduction** (правила записи)

- Возможный формат записи выражения
  x = x op expr
  x = expr op x
  x binop = expr
  x++, ++x, x--, --x
- ***x*** должна быть скалярной переменной
- ***expr*** не должно ссылаться на ***x***
- ***op*** (**operator**) должна быть неперегруженной операцией вида
  +, -, *, /, &, ^, |, &&, ||
- ***binop*** должна быть неперегруженной операцией вида
  +, -, *, /, &, ^, |

Управление областью видимости данных

Параметр **reduction** (пример)

```
#include <omp.h>
main () { /* vector dot product */
 int i, n, chunk;
 float a[100], b[100], result;
 /* Some initializations */
 n = 100; chunk = 10;
 result = 0.0;
 for (i=0; i < n; i++) {
  a[i] = i * 1.0; b[i] = i * 2.0;
 }
 #pragma omp parallel for \
   default(shared) private(i) \
   schedule(static,chunk) \
   reduction(+:result)
 for (i=0; i < n; i++)
  result = result + (a[i] * b[i]);
 printf("Final result= %f\n",result);
}
```

Совместимость директив и их параметров

Clause	Directive					
	PARALLEL	DO/for	SECTIONS	SINGLE	PARALLEL DO/for	PARALLEL SECTIONS
IF	●				●	●
PRIVATE	●	●	●	●	●	●
SHARED	●	●			●	●
DEFAULT	●				●	●
FIRSTPRIVATE	●	●	●	●	●	●
LASTPRIVATE		●	●		●	●
REDUCTION	●	●	●		●	●
COPYIN	●				●	●
SCHEDULE		●			●	
ORDERED		●			●	
NOWAIT		●	●	●		

Функции для контроля/запроса параметров среды исполнения…

- **`void omp_set_num_threads(int num_threads)`**

  Позволяет назначить максимальное число потоков для использования в следующей параллельной области (если это число разрешено менять динамически). Вызывается из последовательной области программы.

- **`int omp_get_max_threads(void)`**

  Возвращает максимальное число потоков.

- **`int omp_get_num_threads(void)`**

  Возвращает фактическое число потоков в параллельной области программы.

Функции для контроля/запроса параметров среды исполнения

Функции для контроля/запроса параметров среды исполнения

- `int omp_get_thread_num(void)`
  Возвращает номер потока.
- `int omp_get_num_procs(void)`
  Возвращает число процессоров, доступных приложению.
- `int omp_in_parallel(void)`
  Возвращает .TRUE., если вызвана из параллельной области программы.
- `void omp_set_dynamic(int dynamic)`
- `int omp_get_dynamic(void)`
  Устанавливает/запрашивает состояние флага, разрешающего динамически изменять число потоков.
- `void omp_get_nested(int nested)`
- `int omp_set_nested(void)`
  Устанавливает/запрашивает состояние флага, разрешающего вложенный параллелизм.

Функции синхронизации…

- В качестве замков используются общие переменные типа **`omp_lock_t`** или **`omp_nestlock_t`**. Данные переменные должны использоваться только как параметры примитивов синхронизации.
- **`void omp_init_lock(omp_lock_t *lock)void omp_nest_init_lock(omp_nest_lock_t *lock)`**

  Инициализирует замок, связанный с переменной
- **`lock.void omp_destroy_lock(omp_lock_t *lock)`**
- **`void omp_destroy_nest__lock(omp_nest_lock_t *lock)`**

  Удаляет замок, связанный с переменной lock.

Функции синхронизации

- `void omp_set_lock(omp_lock_t *lock)`
- `void omp_set_nest__lock(omp_nest_lock_t *lock)`

  Заставляет вызвавший поток дождаться освобождения замка, а затем захватывает его.
- `void omp_unset_lock(omp_lock_t *lock)`
- `void omp_unset_nest__lock(omp_nest_lock_t *lock)`

  Освобождает замок, если он был захвачен потоком ранее.
- `void omp_test_lock(omp_lock_t *lock)`
- `void omp_test_nest__lock(omp_nest_lock_t *lock)`

  Пробует захватить указанный замок. Если это невозможно, возвращает .FALSE.

Переменные среды

OMP_SCHEDULE

Определяет способ распределения итераций в цикле, если в директиве DO использована клауза SCHEDULE(RUNTIME).

OMP_NUM_THREADS

Определяет число нитей для исполнения параллельных областей приложения.

OMP_DYNAMIC

Разрешает или запрещает динамическое изменение числа нитей.

OMP_NESTED

Разрешает или запрещает вложенный параллелизм.

Компилятор с поддержкой OpenMP определяет макрос "**_OPENMP**", который может использоваться для условной компиляции отдельных блоков, характерных для параллельной версии программы

Пример

```
#include <omp.h>
#define THREADNUMS 2
main () { /* вычисление числа π */
 long StepNums = 10000;
 double step, x, pi, sum=0.0;
 int i;
 step = 1.0/StepNums;
 omp_set_num_threads(THREADNUMS);
#pragma omp parallel for reduction(+:sun) private(i,x)
 for (i=0; i<StepNums; i++) {
   x=(i-0.5)*step;
   sum = sum + 4.0/(1.0-x*x);
 }
 pi = step * sum;
}
```

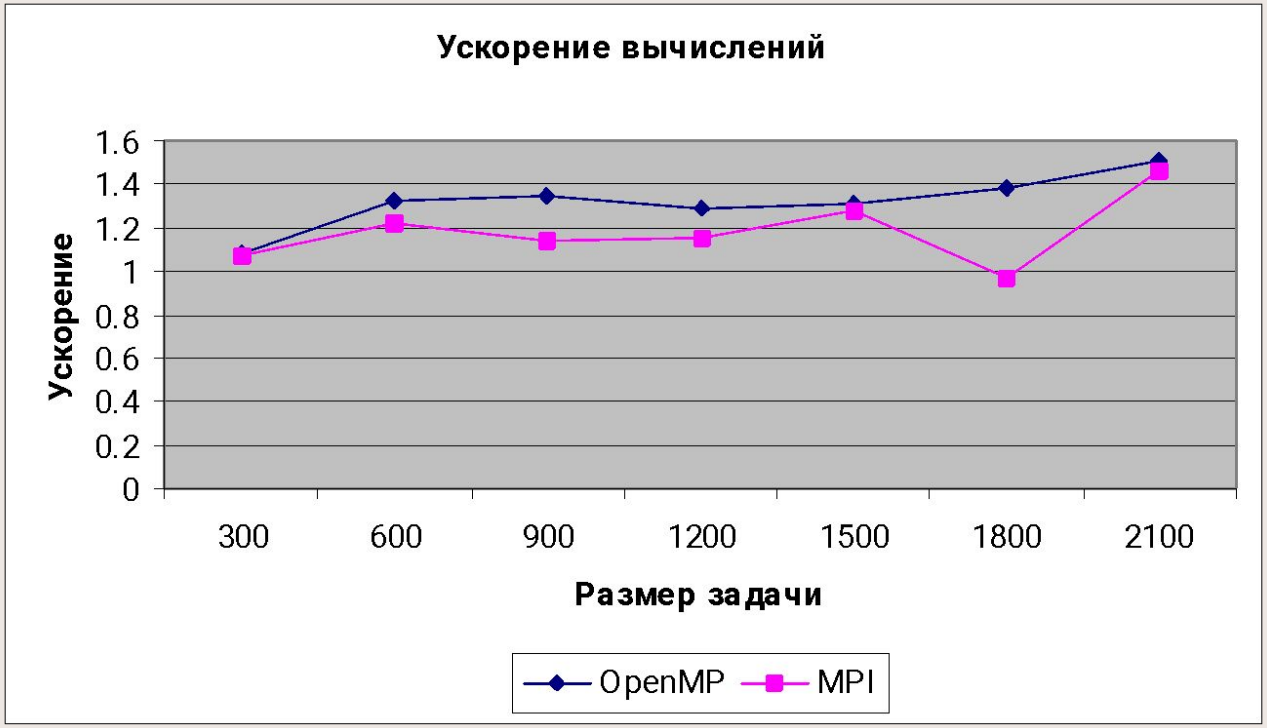
Сравнение технологий MPI и OpenMP для систем с общей памятью…

Сравнение времени выполнения последовательного варианта программы для задачи матричного умножения с вариантами OpenMP и MPI для 2-процессорного сервера

Порядок матрицы: (N)	Время $T_{посл}$ (последовательный алгоритм)	OpenMP		MPI	
		Время $T_{пар}$	Ускорение S	Время T	Ускорение S
300	0.42	0.39	1.08	0.39	1.07
600	4.69	3.55	1.32	3.85	1.22
900	16.20	12.05	1.34	14.17	1.14
1200	38.67	30.00	1.29	33.72	1.15
1500	76.56	58.20	1.32	60.19	1.27
1800	150.08	108.42	1.38	154.73	0.97
2100	258.09	171.75	1.50	177.03	1.46

Сравнение технологий MPI и OpenMP для систем с общей памятью…

Ускорение матричного умножения при использовании параллельных вычислений



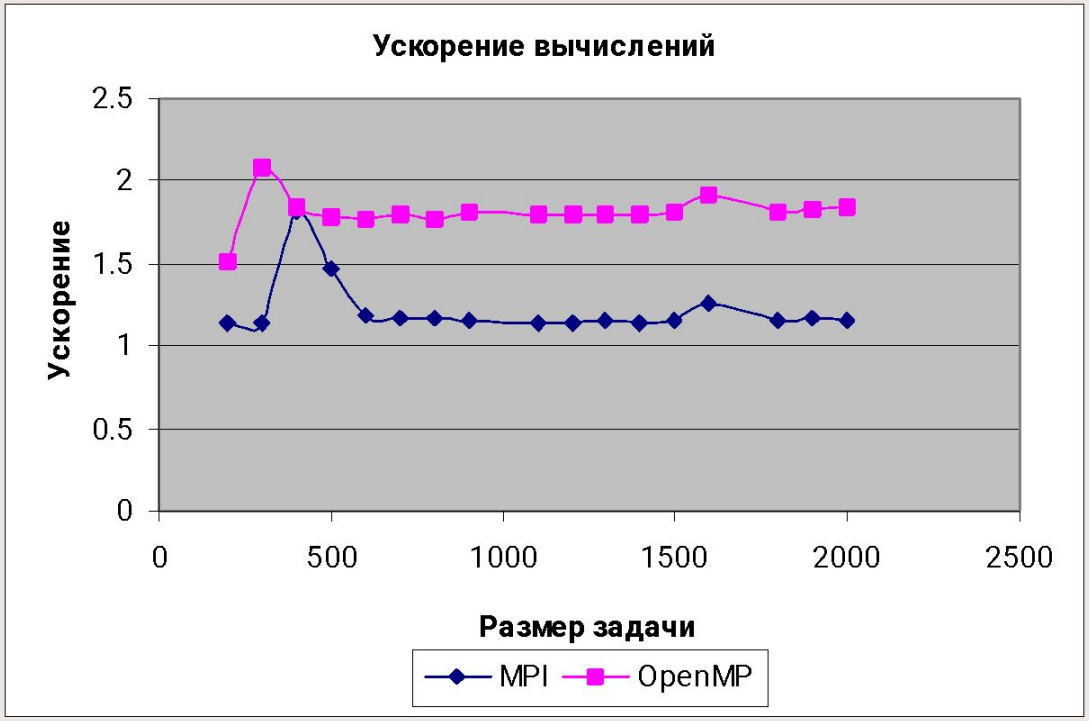
Сравнение технологий MPI и OpenMP для систем с общей памятью…

Сравнение времени выполнения последовательного варианта программы для задачи матричного умножения с вариантами OpenMP и MPI для 4-процессорного сервера

Порядок матрицы: (N)	Время $T_{посл}$ (последовательный алгоритм)	OpenMP		MPI	
		Время $T_{пар}$	Ускорение S	Время T	Ускорение S
300	0.36	0.17	2.09	0.32	1.14
600	6.66	3.78	1.76	5.61	1.19
900	22.92	12.70	1.80	20.01	1.15
1200	54.53	30.30	1.80	48.19	1.13
1500	107.91	59.67	1.81	93.27	1.16
1800	188.61	103.81	1.82	164.45	1.15
2000	262.09	142.73	1.84	226.45	1.16

Сравнение технологий MPI и OpenMP для систем с общей памятью

Ускорение матричного умножения при использовании параллельных вычислений



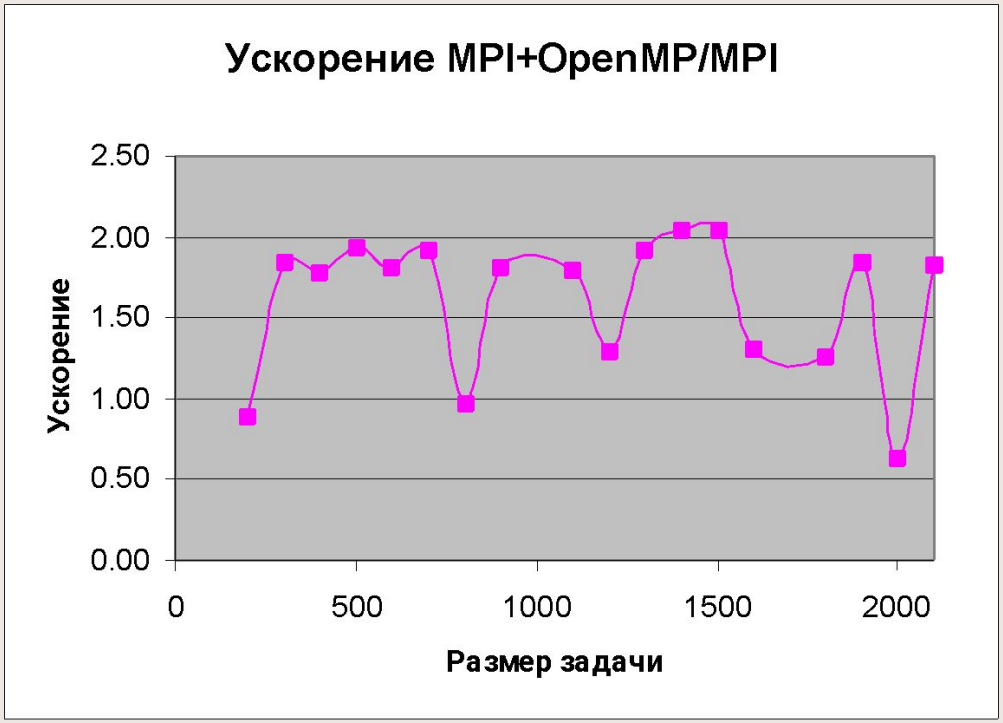
Комбинированная (MPI+OpenMP) технология программирования для систем с общей памятью…

Сравнение времени выполнения MPI и MPI+OpenMP вариантов программы для двух 2-процессорных серверов

Порядок матрицы: (N)	Время $T_{пар}$ (сек)		Ускорение S
	MPI	MPI+OpenMP	
300	0.52	0.28	1.84
600	3.95	2.18	1.82
900	14.03	7.75	1.81
1200	37.36	29.08	1.28
1500	65.39	32.03	2.04
1800	112.75	89.29	1.26
2100	185.85	101.97	1.82

Комбинированная (MPI+OpenMP) технология программирования для систем с общей памятью

Ускорение матричного умножения при использовании смешанного MPI+OpenMP варианта параллельной программы



Реализации OpenMP

1) **Silicon Graphics**. Fortran 77/90 (IRIX), планируется поддержка OpenMP для C/C++.
2) **Compaq/DEC**. **DIGITAL Fortran**
3) **Kuck & Associates** (KAI). Fortran, C/C++ (Unix, Windows)
4) **Portland Group** (PGI). Fortran и C/C++ для Windows NT, Linux, Solaris (x86).
5) **OdinMP**. OpenMP-препроцессор для языка C, генерация программы в стандарте POSIX threads.
6) **Sun**. Планируется поддержка OpenMP
7) **Pacific-Sierra Research** предлагает распараллеливающие препроцессоры **VAST/Parallel** для Fortran и C, которые обеспечивают автоматическое распознавание параллелизма в программах и выполнение необходимой трансформации программ путем добавления соответствующих директив OpenMP.
8) **Intel**. Fortran, C/C++ (Unix, Windows)

Информационные ресурсы

- www.openmp.org
- Что такое OpenMP - http://parallel.ru/tech/tech_dev/openmp.html
- OpenMP C/C++ specification v1.0 http:// www.openmp.org
- Introduction to OpenMP - www.llnl.gov/computing/tutorials/workshops/workshop/openMP/MAIN.html
- Chandra, R., Menon, R., Dagum, L., Kohr, D., Maydan, D., McDonald, J. (2000). *Parallel Programming in OpenMP.* Morgan Kaufmann Publishers.

Вопросы для обсуждения

- Методы синхронизации обработки данных в OpenMP
- Программирование с использованием библиотеки функций OpenMP

Задания для самостоятельной работы

- Разработка параллельных методов для задач линейной алгебры при использовании интерфейса OpenMP

Заключение

- Методы синхронизации обработки данных
- Библиотека функций OpenMP
- Переменные среды окружения