VI-SEEM project

Обучение в ИИКТ-БАН, 07 Юли 2016: Въведение в паралелното програмиране. Паралелни пресмятания с MPI и OpenMP

https://vi-seem.eu

доц. д-р Гергана Бенчева
ИИКТ - БАН
gery@parallel.bas.bg

The VI-SEEM project initiative is co-funded by the European Commission under the H2020 Research Infrastructures contract no. **675121**

Съдържание

- Мотивация и терминология
- Паралелни компютърни архитектури
- Модели за паралелно програмиране
- Подход за конструиране на ПА
- Оценка на ефективността
- Средства за реализация
  - OpenMP
  - MPI
- Източници на информация

МОТИВАЦИЯ И ТЕРМИНОЛОГИЯ

Пример за паралелизъм

Паралелна обработка на информация – извършване на нещата едновременно с цел по-бързо и по-ефективно получаване на резултат.

Един пример: Ако 1 човек измива 1 кола за 1 час, дали

- 2 души ще измият 2 коли за 1 час?
- 2 души ще измият 1 кола за 1/2 час?
- 10 души ще измият 10 коли за 1 час?
- 10 души ще измият 1 кола за 1/10 час?

Отговор: зависи от това как си разделят работата, могат ли да работят едновременно, има ли достатъчно място и/или мръсни коли, за да са заети всички.

Извод: Не винаги ако удвоим броя на ”работниците” ще удвоим ефективността

Видове паралелна обработка:

- ”в дълбочина”
- ”на ширина”
- ”реална”
- ”симулирана”

Паралелни пресмятания

Паралелно пресмятане **(parallel computing)** е едновременно използване на множество компютърни ресурси за решаване на определена изчислителна задача:

Компютърните ресурси могат да бъдат:

- един компютър с множество процесори;
- произволен брой компютри свързани в мрежа;
- комбинация от двете.

Изчислителната задача обикновено дава възможност:

- за разделяне на дискретни части, всяка от които представлява серия от инструкции;
- инструкциите от всяка част да се изпълняват едновременно от различни процесори;
- за изпълнение на многократни програмни инструкции в произволен момент от време;
- да бъде решена за по-малко време с множество компютърни ресурси, отколкото с един изчислителен ресурс (един процесор).

Използване на паралелни изчисления

- Защо?
  - Пестят време и/или пари
  - Решават се по-големи задачи
  - Осигуряват едновременно изпълнение (concurrency)
  - Възможност за използване на отдалечени ресурси
  - Ограничения в бързодействието на последователните компютри
- От кого и за какво?
  - top500.org предлага статистика за потребителите на паралелни изчислителни системи
  - Повече от 5 групи потребители, повече от 30 вида приложения
- Бъдещето
  - по-бързите мрежи, разпределените системи и мулти-процесорните компютърни архитектури показват, че паралелизмът е бъдещето на компютърните приложения

ПАРАЛЕЛНИ КОМПЮТЪРНИ АРХИТЕКТУРИ

Класификация

- Класификация на Флин (1966): инструкции-данни

	Single Instruction	Multiple Instruction
Single Data	SISD	MISD
Multiple Data	SIMD	MIMD

- Класификация според нивото на паралелизъм на хардуера
  - Многоядрени пресмятания (Multicore computing)
  - Симетрични мултипроцесорни изчисления (Symmetric multiprocessing)
  - Разпределени изчисления (Distributed computing)
    - Клъстерни изчисления (Cluster computing)
    - Масивно паралелни изчисления (Massive parallel processing)
    - Грид изчисления (Grid computing)
  - Специализирани паралелни компютри
    - Векторни процесори (Vector processor)
    - Изчисления върху GPU (General-purpose computing on Graphics Processing Units)
    - Изчисления върху MIC (Many-integrated core)

Терминология

- Раздробеност (Granularity) – качествена мярка за отношението на изчисления към комуникации
  - Едро/грубо (coarse) – извършват се относително големи количества изчисления между комуникационни събития
  - Фино/дребно (fine) – извършват се относително малки количества изчисления между комуникационни събития
  - Независими задания (embarassingly parallel) – едновременно решаване на множество подобни, но независими задачи; малка до никаква необходимост от координиране между заданията.
- Паралелни ”забавяния” (Parallel overhead):

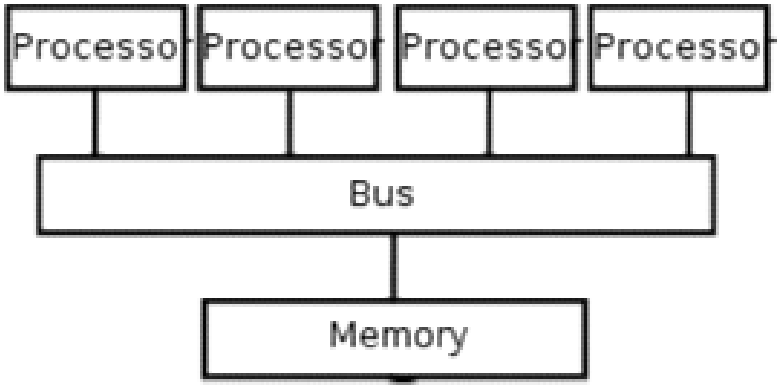
времето необходимо за координиране на паралелни задания като противоположност на полезната работа

Терминология (продълж.)

- Скалируемост – отнася се до възможността на паралелната система (софтуер и/или хардуер) да демонстрира пропорционално увеличаване на паралелното ускорение с добавяне на повече процесори. Фактори, които влияят на скалируемостта:
  - Хардуер – в частност ширината на лентата за комуникации между памет и процесор и тези по мрежа;
  - Прилагания алгоритъм
  - Паралелните забавяния
  - Характеристиките на вашите специфични приложение и програма
- Топология на свързване – физическата връзка между процесорите: звезда, пръстен, мрежа, хиперкуб ...

Обща памет - характеристики

- Всички процесори имат достъп до цялата памет като глобално адресно пространство.
- Няколко процесора могат да работят независимо един от друг, като споделят едни и същи ресурси памет.
- Промяна на стойност в паметта, направена от един процесор, е видима за всички други процесори.
- Два основни класа според времената за достъп до паметта на различните процесори – UMA (Uniform Memory Access) и NUMA (Non-Uniform Memory Access).
  - UMA – SMP, идентични процесори, равноправен достъп и време за достъп до паметта;
  - NUMA – няколко физически свързани SMP, не всички процесори имат равно време за достъп до всяка от паметите.



Обща памет – предимства и недостатъци

Предимства

- Глобалното адресно пространство предоставя удобни за потребителя средства за работа с паметта в програмни реализации.
- Времето за споделяне на данни между отделни паралелни задания е бързо и равномерно, благодарение на физическата близост на паметта и процесорите.

Недостатъци

- Основен недостатък е липсата на скалируемост за отношението на обем памет и брой използвани процесори. Добавянето на повече процесори може да доведе до забавяне, породено от геометрично увеличение на трафика по пътя между паметта и процесорите, а при системи със съгласуваност на кеша, увеличение на трафика асоцииран с управлението на връзките между кеша и паметта
- Програмистът има отговорността така да синхронизира изпълнението на заданията, че да гарантира ”коректни” резултати при заявки към паметта.
- Цена: все по-трудно и скъпо е да се проектират и произвеждат машини с обща памет с все по-голям брой процесори.

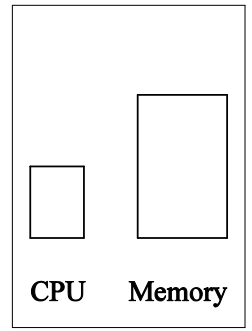
Разпределена памет - характеристики

- За осъществяване на връзка между процесорите и паметта е необходима комуникационна мрежа.
- Процесорите имат собствена локална памет. Адресните пространства за различни процесори не съвпадат и не съществува понятие за общо адресно пространсвто за всички процесори.
- Тъй като всички процесори имат собствена памет, те работят независимо. Промени в локалната памет на един процесор не се отразяват в паметта на другите. Поради тази причина понятието за съгласуваност на кеша не е приложимо.
- При необходимост от достъп на един процесор до данните в паметта на друг, програмистът явно трябва да дефинира как и кога да се разменят тези данни, както и да синхронизира работата на съответните паралелни задания.
- Средствата за осигуряване на трансфера на данни са много разнообразни, като могат да се използват дори обикновени етернет мрежи.

Разпределена памет – предимства и недостатъци

Предимства

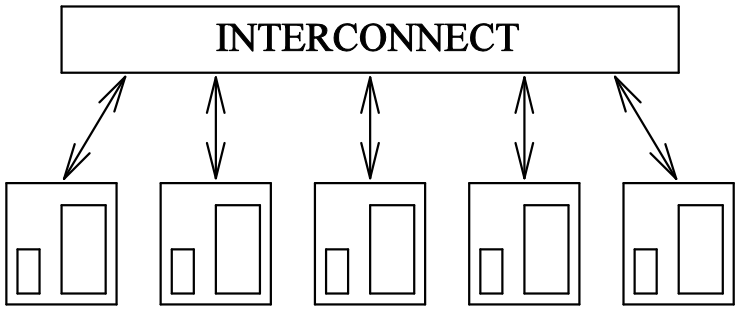
- Паметта е скалируема по отношение на броя процесори.
  При увеличаване на този брой, обема на паметта се увеличава пропорционално.
- Всеки процесор осъществява бърз и непрекъснат достъп до локалната си памет, без забавяне поради необходимост от съгласуване на кеша.
- Ценова ефективност: могат да се използват широко достъпни процесори и мрежи.



Фон Нойман

Недостатъци

- Програмистът е отговорен за много от детайлите при осъществяване на комуникацията между процесорите.
- Може да се окаже трудно директното изобразяване на съществуващи структури от данни, проектирани за глобална памет, в програми използващи разпределена памет.
- Неравномерни времена за достъп до паметта.



Разпределена памет

Хибридни архитектури с обща и разпределена памет

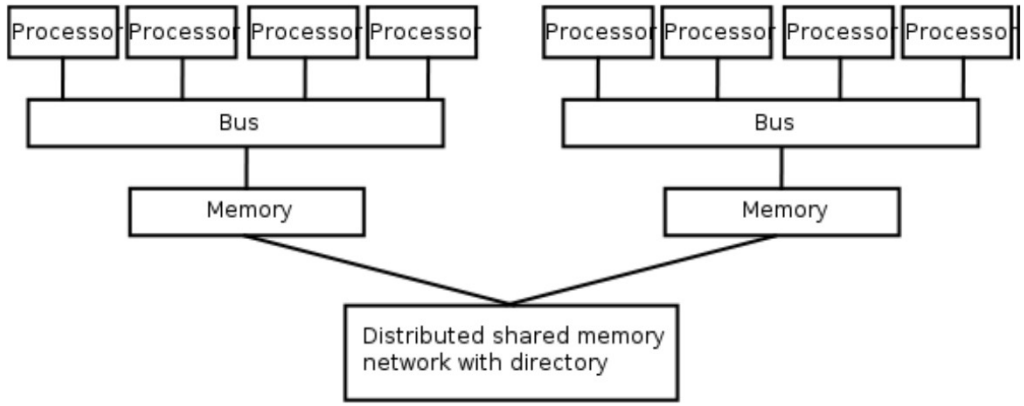
Общи характеристики

- Най-големите и бързи компютри в света в днешно време съчетават архитектури с обща и разпределена памет.
- Компонентът с обща памет най-често е SMP машина със съгласуван кеш. Процесорите в една SMP третират паметта на машината като глобална.
- Компонентът с разпределена памет се реализира чрез свързване в мрежа на няколко SMP машини. Отделните SMP виждат само своята памет, не и паметта на другите машини в мрежата. За прехвърляне на данни между тях се използват комуникации през мрежата.
- Настоящите тенденции предполагат, че в близко бъдеще хибридните архитектури ще продължават да преобладават и да увеличават списъка на най-бързите и мощни компютърни системи.

Предимства и недостатъци

- общите за архитектури с

обща и разпределена памет



МОДЕЛИ ЗА ПАРАЛЕЛНО ПРОГРАМИРАНЕ

Модели за паралелно програмиране

- Съществуват като абстракция над хардуера и организацията на паметта;
- НЕ са специфични за определен тип машина или архитектура, а теоретично всяка от тях може да се приложи върху произволен хардуер (виртуална обща памет, предаване на съобщения за система с обща памет)
- Разпространени са няколко модела:
  - Обща памет – заданията споделят общо адресно пространство; асинхронно чете и писане; контрол на достъпа с механизми като заключване и семафори;

    + няма собственост върху данните и нужда от определяне на комуникации;

    - става трудно да се разбере и управлява локалността на данните;
  - Нишки – POSSIX Threads, Open MP
  - Предаване на съобщения – MPI
  - Паралелни данни – извършване на операции върху набор от данни (масив или куб); всяко задание работи върху различна част от тях; F90, F95, HPF
  - Хибрид – комбинация от вече споменатите.

Нишки

Принципи

- ОС стартира главната програма a.out и зарежда необходимите ресурси
- a.out изпълнява последователна работа, след което генерира няколко задания (нишки), които могат да се стартират от ОС едновременно
- Всяка нишка има своя локална памет, но споделя и общите ресурси на a.out и печели от достъпа до общата памет, осигурен от a.out.
- Произволна нишка може да изпълни произволна подпрограма по едно и също време с останалите нишки.
- Нишките комуникират помежду си чрез глобалната памет. Изисква синхронизация, за да се гарантира, че не повече от една нишка променя даден адрес в глобалната памет в произволен момент.
- Нишките могат да идват и да си отиват, но a.out остава, за да сигури необходимите споделени ресурси до завършване на приложението.

Реализация:

- Обикновено включват библиотеки от подпрограми извиквани в рамките на паралелния код и/или набор от директиви на компилатора вградени в последователния или в паралелния код. Програмистът определя паралелизма.
- Стандартизация – POSSIX Threads (Pthreads) и Open MP.

Предаване на съобщения

Принципи

- Множество от няколко задания, които имат тяхна локална памет.
- Заданията могат да се намират върху една и съща физическа машина или върху произволен брой машини.
- Ако една променлива е декларирана във всяко задание на програма с p задания, тогава има p различни променливи с едно и също име, но с евентуално различни стойности.
- Ако изпълнението на функция от дадено задание изисква данни от друго задание, данните се изпращат чрез оператор за предаване на съобщения и в двете (изпращащо и получаващо) задания.
- Върху почти всички архитектури, предаването на съобщения е много по-бавно от аритметиката с плаваща запетая.

Реализация

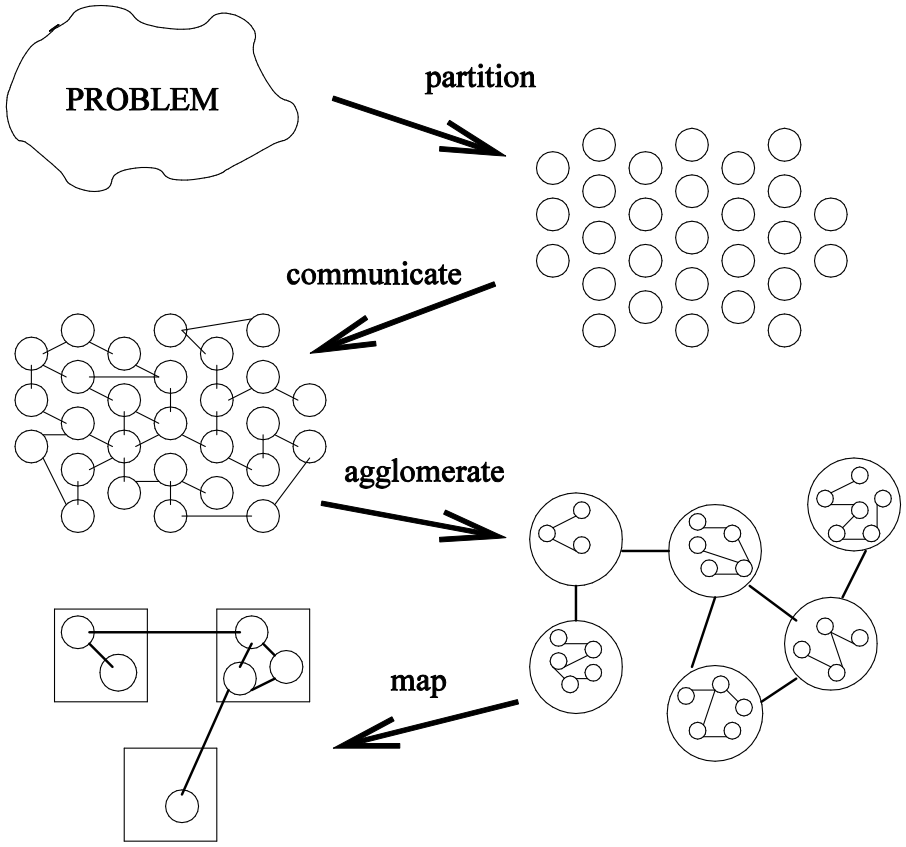
- Включват библиотеки от подпрограми извиквани в рамките на паралелния код. Програмистът определя паралелизма.
- Стандартизация – Message Passing Interface (MPI).
- За архитектури с обща памет, реализациите на MPI обикновено не използват мрежа за комуникациите между заданията. Вместо това се използва общата памет за по-добра производителност.

ПОДХОД ЗА КОНСТРУИРАНЕ НА ПА

Основни етапи

Преди да започнем: може ли задачата да се реши паралелно?

1. Разделяне
2. Комуникации
3. Агломерация
4. Изобразяване



1 и 2 – не зависят от машината

3 и 4 – за конкретна паралелна архитектура и производителност

I. Foster, Designing and building parallel programs, Addison-Wessley, 1995.

Разделяне

Изчисленията, които трябва да се извършат и данните, с които се оперира се разделят на малки задания (по възможност – непресичащи се). Практически въпроси, като брой процесори в достъпния компютър се игнорират и вниманието се насочва върху откриване на възможности за паралелно изпълнение.

а) разделяне на областта (domain decomposition) – първо се определя подходящо разделяне на данните на задачата и след това се свързват изчисленията с данните.

б) функционално разделяне (functional decomposition) – първо се разделят изчисленията и след това се разглеждат данните.

Списък за проверка

1. Дефинирани ли са поне с порядък повече задания отколкото са процесорите на достъпния компютър? **Ако не** – имате по-малка гъвкавост в следващите етапи на схемата.

2. Избягват ли се излишни пресмятания и място за съхранение на данни? **Ако не** – крайният алгоритъм може да не е достатъчно ефективен за големи задачи.

3. Задчите със сравнима големина ли са? **Ако не** – може да е трудно да се осигури равно количество работа за всички процесори.

4. Зависи ли броят на подзаданията от размерността на задачата? В идеалния случай увеличаването на размерността на задачата води до увеличаване на броя на заданията, а не на тяхната размерност. **Ако не** – ПА няма да може да решава по-големи задачи, когато са достъпни повече процесори.

5. Определени ли са няколко различни разделяния? Дават възможност за увеличаване на гъвкавостта на следващите етапи.

Комуникации

Vi-SEEM

Определят се необходимите комуникации за съгласуване на изпълнението на заданията и се дефинират подходящи комуникационни структури. Целта е да се разпределят операциите за комуникация (без да се въвеждат ненужни такива) върху много задания и да се организират по начин, който позволява едновременно изпълнение.

- **локални/глобални** – всяко задание комуникира с малко/голямо множество други задания;
- **структурирани/неструктурирани** – задание и неговите съседи образуват регулярна структура (напр. дърво или мрежа)/произволен граф;
- **статични/динамични** – вида на комуникационните структури не се променя с времето или се определя според изчислените данни на отделни етапи от изпълнението на алгоритъма;
- **синхронни/асинхронни** – производителят и потребителят на данните си съдействат/не си съдействат в операциите за обмен.

Списък за проверка

1. Дали всички задания изпълняват приблизително равен брой комуникации? **Ако не** – неефективна конструкция. Преразгледайте възможността за по-равномерно разпределение на операциите за комуникация. Напр., ако структура от данни, до която често има достъп се намира в едно задание, разгледайте възможността за нейното разделяне или дублиране.

2. Дали всяко задание комуникира с малък брой съседи? **Ако не** – оценете възможността тази глобална комуникация да се преформулира в термините на локална комуникационна структура.

3. Възможно ли е операциите за комуникация да се извършват едновременно? **Ако не** – вашият алгоритъм изглежда е неефективен и без ускорение. Използвайте подхода ”разделяй и владей” (”divide-and-conquer”).

4. Възможно ли е изчисленията свързани с различни задания да се извършват едновременно? **Ако не** – вашият алгоритъм изглежда е еефективен и без ускорение. Възможно ли и пренареждане на операциите за комуникации и изчисления?

Агломерация

Заданията и комуникационните структури от първите два етапа се оценяват по отношение на конкретната производителност и реализацията. Целта е да се получи ефективен алгоритъм за определен клас паралелни компютри. Ако е необходимо, заданията се групират в по-големи и се дублират данни и/или изчисления, за да се подобри производителността или за да се намали цената за разработване.

Списък за проверка

1. Дали агломерацията намалява цената на комуникациите, чрез увеличаване на локалността?
2. Дали ползата от евентуално дублиране на изчисления е по-голяма от цената му за множество различни размерности на задачата и брой процесори?
3. Дали евентуалното дублиране на данни не излага на риск ефективността чрез намаляване на диапазона на размерностите на задачите или броя на процесорите?
4. Агломерацията доведе ли до задания с приблизително равностойни цени на изчисленията и комуникациите?
5. Дали броя на заданията все още зависи от размерността на задачата?

Изобразяване

Всяко задание се свързва с процесор по такъв начин, че да се удовлетворят конкуриращите се условия за максимално използване на процесорите и минимизиране на комуникациите. Изобразяването може да бъде зададено статично или да се определя по време на изпълнението чрез алгоритми за балансиране на натоварването. (Върху системи с обща памет това се прави автоматично.)

Баланс между две стратегии:

1. Увеличаване на паралелизма – задания, които могат да се изпълняват едновременно се разполагат върху различни процесори.
2. Увеличаване на локалността – задания, които си комуникират често се разполагат в един и същ процесор.

ОЦЕНКА НА ЕФЕКТИВНОСТТА

Оценка на ефективността

- **Ускорение:** $S_P = \frac{T_1}{T_P}$, $S_P \leq P$; **Ефективност:** $E_P = \frac{S_P}{P} = \frac{T_1}{P.T_P}$, $E_P \leq 1$

- Наблюдавани ускорение и ефективност – T_1 и T_P са измерените (wall-clock) времена съответно за последователно и паралелно изпълнение на програмата.

- Теоретична оценка на T_P при MIMD с разпределена памет и P процесора

$$T_P = T_a + T_{com}, \quad T_a = M * t_a, \quad T_{com} = c_1 * t_s + c_2 * t_w$$

t_a е усреднено време за извършване на 1 ар. оп. от 1 процесор

t_s е времето за стартиране на комуникация

t_w е времето за изпращане на едно число до съседен процесор

c_1 и c_2 са константи или функции на размерността на задачата, броя на процесорите, разстоянието между тях

СРЕДСТВА ЗА РЕАЛИЗАЦИЯ

Средства за реализация – Open MP

Open MP: http://www.openmp.org

- е преносима реализация на модела за паралелно програмиране чрез нишки.
- Включва три основни компоненти:
  - Директиви на компилатора
  - Библиотечни функции по време на изпълнение
  - Променливи на средата
- Има реализации на Fortran77 и C/C++
- Може да се използва върху множество платформи, вкл. UNIX/Linux и Windows

Open MP не е:

- замислен за паралелни системи с разпределена памет
- задължително реализиран по еднакъв начин от всички производители
- гарантирано, че използва максимално ефективно обща памет
- не гарантира синхронизиран вход или изход от един и същи файл – това трябва да се осигури от програмиста

Компоненти на OpenMP

- Включване на библитеката #include <omp.h>
- Директиви на компилатора

```
#pragma omp parallel
{       //code segment
}
```

```
#pragma omp for
for()
{ … }
```

контрол на данни и среда чрез private, firstprivate, lastprivate, num_threads

- Библиотечни функции

int omp_get_num_threads(void)	текущ брой използвани нишки
int omp_set_num_threads(int NumThreads)	преди влизането в паралелна част указва с колко нишки да се изпълни
int omp_get_thread_num(void)	връща номера на текущата нишка
int omp_get_num_procs(void)	връща броя на достъпните ядра

- Променливи на средата
  - OMP_SCHEDULE — контролира разпределянето на изпълнението на цикъл между нишките
  - OMP_NUM_THREADS — дефинира броя нишки в паралелния участък

Средства за реализация – MPI

MPI (Message Passing Interface) е преносим стандарт за предаване на съобщения, който подпомага разработването на паралелни приложения и библиотеки

- достъпен върху UNIX, Linux, Windows и др.
- скрива разликите в компютърните архитектури
- може да се използва и върху системи с обща памет, както и върху хибридни архитектури.

Различни реализации:

- по отношение на езика за програмиране:
  - официален стандарт – поддръжка на Fortran и C/C++
  - неофициални реализации за Java, Python, Matlab и др.
- безплатни реализации:
  - MPICH: www.mcs.anl.gov/research/projects/mpi/mpich2
  - Open MPI: http://www.open-mpi.org/
  - LAM/MPI: http://www.lam-mpi.org/
- платени реализации
  - Intel MPI: http://software.intel.com/en-us/intel-mpi-library
  - HP MPI: http://www.hp.com/go/mpi

Основни функции

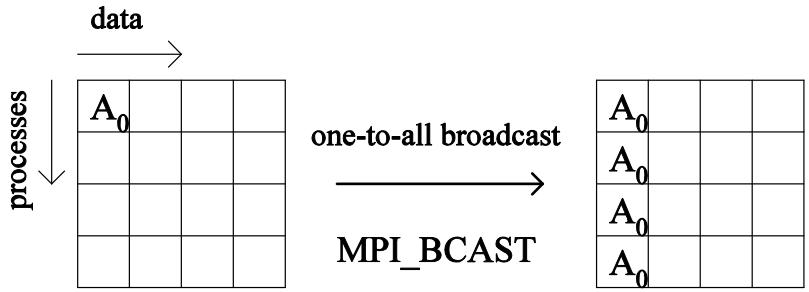
- Включване на библитеката #include <mpi.h>
- Управление на средата: Всички MPI функции връщат MPI_SUCCESS ако операцията е успешна
  - MPI_Init(&argc, &argv) Започва MPI изчисление
  - MPI_Comm_size(comm, &size) Определя броя на процесите
  - MPI_Comm_rank(comm, &rank) Определя идентификатор на викащия процес
  - MPI_COMM_WORLD Комуникатор (всички процеси)
  - MPI_Finalize() Прекратява изчисление
  - MPI_Wtime() Измерва на времето в секунди от фиксиран момент в миналото
- Комуникации между два процесора
  - MPI_Send(buffer,count,type,dest,tag,comm) Блок. изпращане
  - MPI_Isend(buffer,count,type,dest,tag,comm,request) Неблок. изпращане
  - MPI_Recv(buffer,count,type,source,tag,comm,status) Блок. получаване
  - MPI_Irecv(buffer,count,type,source,tag,comm,request) Неблок. получаване

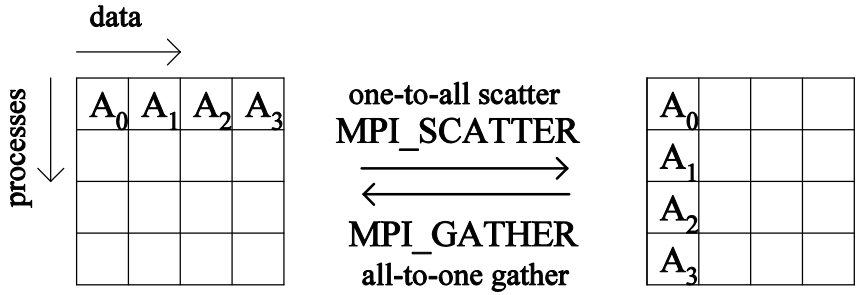
Основни типове в MPI и съответните им в C

MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED_SHORT	unsigned short int
MPI_UNSIGNED	unsigned int
MPI_UNSIGNED_LONG	unsigned long int
MPI_FLOAT	float
MPI_DOUBLE	double
MPI_LONG_DOUBLE	long double
MPI_BYTE	8 binary digits
MPI_PACKED	data packed or unpacked with MPI_Pack()/MPI_Unpack

Има възможност за дефиниране на допълнителни типове.

Глобални комуникации - I





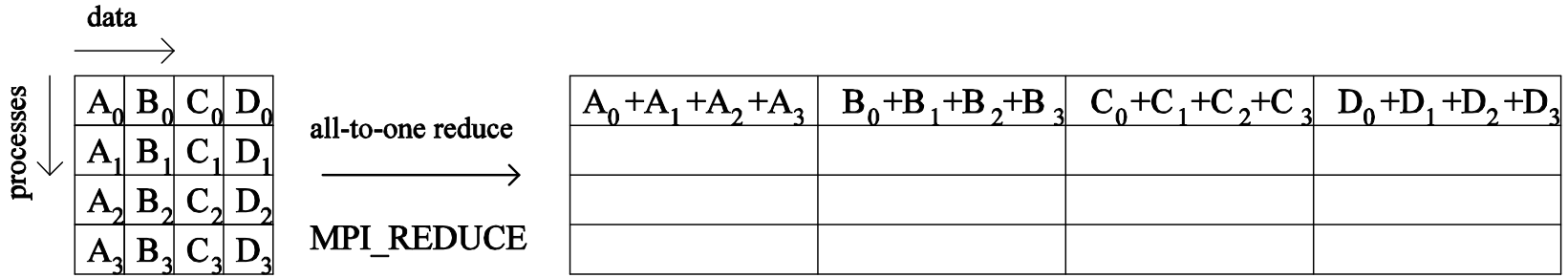
- Предаване на M числа от един процесор до всички останали за време b(P, M) (one_to_all broadcast);
- Един от процесорите изпраща до всеки от останалите по един пакет с данни, съдържащ M числа, за време s(P, M) (one_to_all scatter). В общия случай данните в пакетите за отделните процесори са различни. Дуалната операция (all_to_one gather) – събиране на P такива пакета от всички останали в един процесор изисква същото време s(P, M);

MPI_Bcast(&buffer,count,datatype,root,comm)

MPI_Scatter(&sendbuf,sendcnt,sendtype,&recvbuf,recvcnt,recvtype,root,comm)

MPI_Gather(&sendbuf,sendcnt,sendtype,&recvbuf,recvcount,recvtype,root,comm)

Глобални комуникации - II



- Всеки процесор има пакет от M числа (в общия случай различен от пакетите на останалите процесори). След приключване на операцията (за време r(P, M)) един от процесорите притежава пакет, който е сума, разлика, максимум или друга операция приложена поелементно върху изходните пакети (all_to_one reduce). Схемата за извършването ´и е обратна на one_to_all broadcast и следователно е в сила r(P, M) = b(P, M). Има вариант на тази операция, при който резултата е във всички процесори: ar(P, M) = 2 ∗ b(P, M) (all_to_all reduce).

MPI_Reduce(&sendbuf,&recvbuf,count,datatype,op,root,comm)

MPI_Allreduce(&sendbuf,&recvbuf,count,datatype,op,comm)

MPI_Barrier(comm)

ИЗТОЧНИЦИ НА ИНФОРМАЦИЯ

Източници на информация

- Книги
  - I. Foster, Designing and building parallel programs, Addison-Wessley, 1995.
    http://www.mcs.anl.gov/~itf/dbpp/
  - V. Kumar, A. Grama, A. Gupta, G. Karypis, Introduction to parallel computing: design and analysis of algorithms, Addison-Wesley, 1994, 2003.
    http://www-users.cs.umn.edu/~karypis/parbook/
- WWW – стандарти:
  - MPI: http://www.mpi-forum.org
  - MPICH: www.mcs.anl.gov/research/projects/mpi/mpich2
  - Open MP: http://openmp.org/wp/
- WWW – обучение:
  - https://computing.llnl.gov/tutorials
  - http://en.wikipedia.org/wiki/Parallel_computing

Търсене във WWW за ”parallel programming” или ”parallel computing”