Интернет Университет Суперкомпьютерных технологий

Введение

Учебный курс

Параллельное программирование с OpenMP

Бахтин В.А., кандидат физ.-мат. наук,
заведующий сектором,
Институт прикладной математики им. М.В.Келдыша РАН

Содержание

- Тенденции развития современных процессоров
- Существующие подходы для создания параллельных программ
- Основные возможности OpenMP
- Литература

Тенденции развития современных процессоров

В течение нескольких десятилетий развитие ЭВМ сопровождалось удвоением их быстродействия каждые 1.5-2 года. Это обеспечивалось и повышением тактовой частоты и совершенствованием архитектуры (параллельное и конвейерное выполнение команд).

Узким местом стала оперативная память. Знаменитый закон Мура, так хорошо работающий для процессоров, совершенно не применим для памяти, где скорости доступа удваиваются в лучшем случае каждые 6 лет.

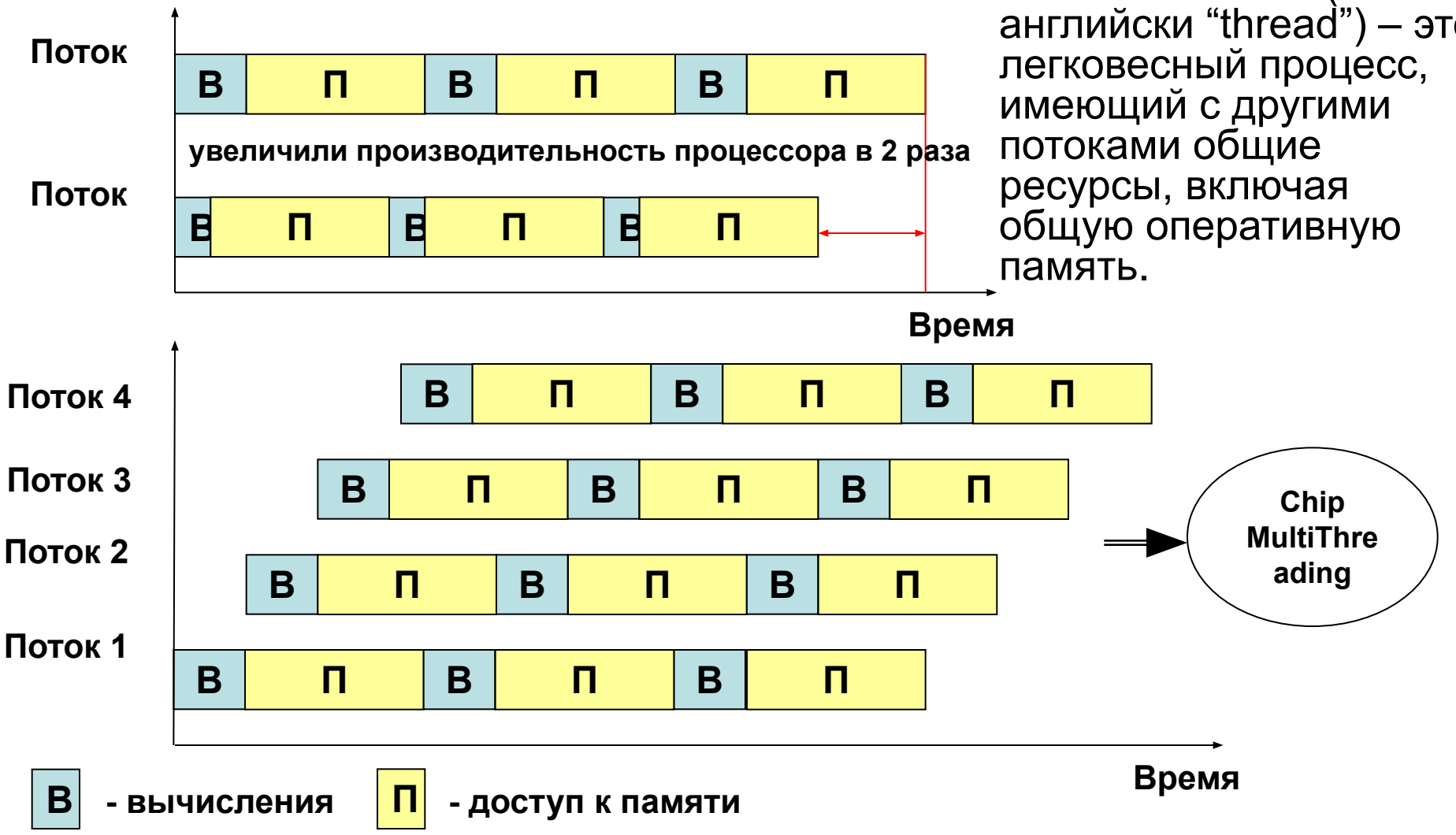
Совершенствовались системы кэш-памяти, увеличивался объем, усложнялись алгоритмы ее использования.

Для процессора Intel Itanium:

- Latency to L1: 1-2 cycles
- Latency to L2: 5 - 7 cycles
- Latency to L3: 12 - 21 cycles
- Latency to memory: 180 – 225 cycles

Важным параметром становится - **GUPS** (Giga Updates Per Second)

Тенденции развития современных процессоров



Поток или **нить** (по-английски “thread”) – это легковесный процесс, имеющий с другими потоками общие ресурсы, включая общую оперативную память.

Тенденции развития современных процессоров

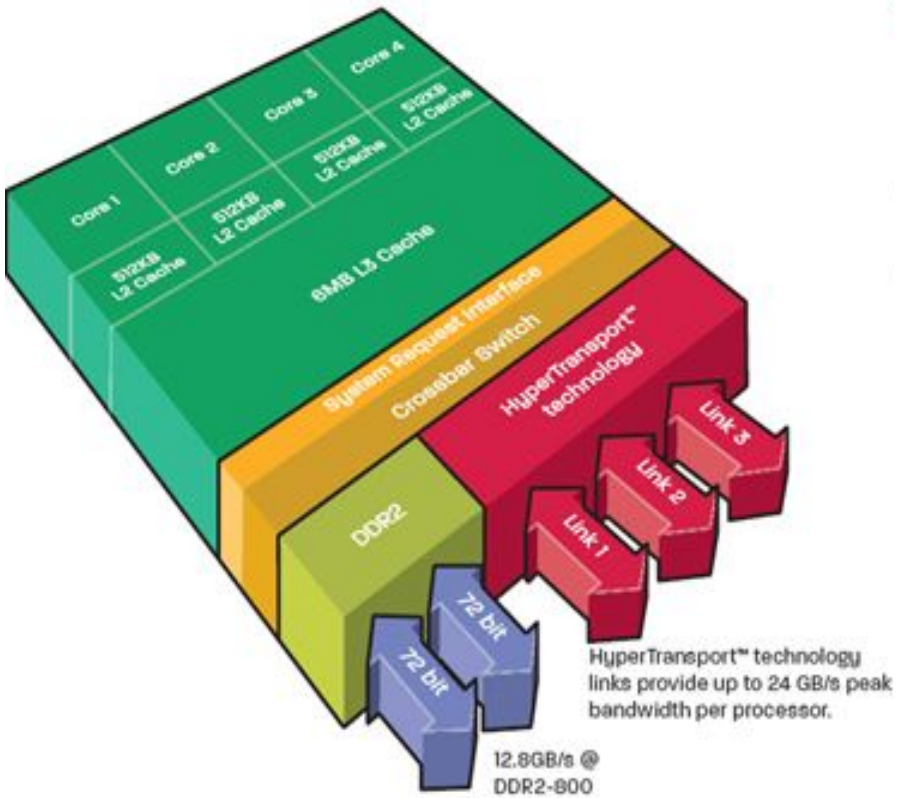
Суперкомпьютер СКИФ МГУ «Чебышев»

- Пиковая производительность - 60 TFlop/s
- Число процессоров/ядер в системе - 1250 / 5000
- Производительность на Linpack - 47.04 TFlop/s (78.4% от пиковой)
- Номинальное энергопотребление компьютера - **330 кВт**
- Энергопотребление комплекса - **720 кВт**

Важным параметром становится – **Power Efficency (Megaflops/watt)**

Как добиться максимальной производительности на Ватт => Chip MultiProcessing, многоядерность.

Тенденции развития современных процессоров



Quad-Core AMD Opteron

- 4 ядра
- встроенный контроллер памяти (2 канала памяти DDR2 800 МГц)
- 3 канала «точка-точка» с использованием HyperTransort

Тенденции развития современных процессоров

Intel Core i7 (архитектура Nehalem)

- 4 ядра
- 8 потоков с технологией Intel Hyper-Threading
- 8 МБ кэш-памяти Intel Smart Cache
- встроенный контроллер памяти (3 канала памяти DDR3 1066 МГц)
- технология Intel QuickPath Interconnect

Тенденции развития современных процессоров

SUN UltraSPARC T2 Processor (Niagara 2)

- 8 ядер
- 64 потоков
- 4 контроллера памяти
- потребляемая мощность – 60-123Ватт
- встроенный котроллер 2x10 Gigabit Ethernet

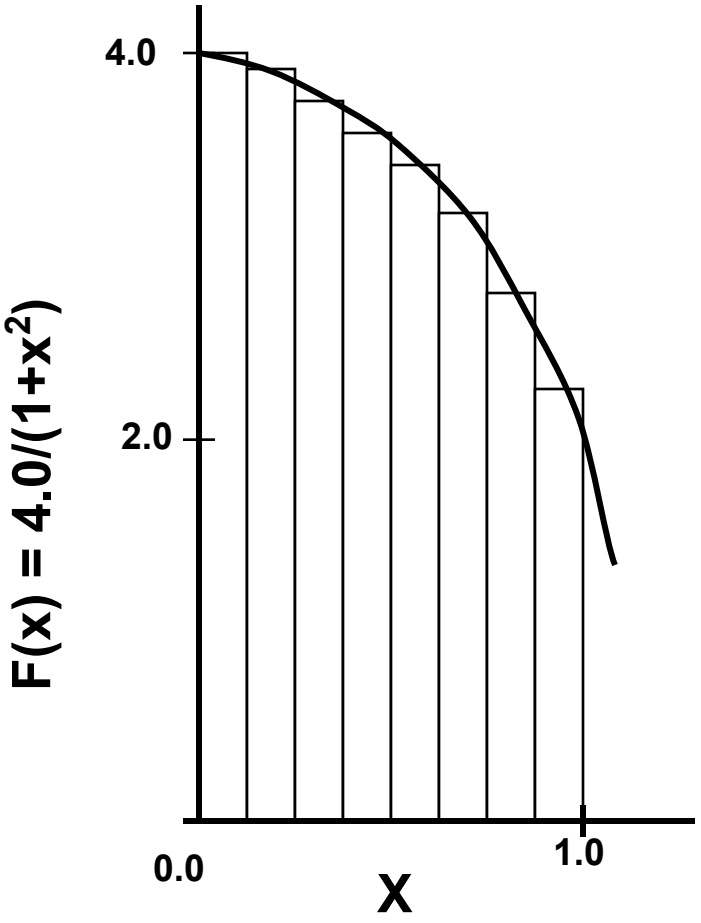
Тенденции развития современных процессоров

- Темпы уменьшения латентности памяти гораздо ниже темпов ускорения процессоров + прогресс в технологии изготовления кристаллов => CMT (Chip MultiThreading)
- Опережающий рост потребления энергии при росте тактовой частоты + прогресс в технологии изготовления кристаллов => CMP (Chip MultiProcessing, многоядерность)
- И то и другое требует более глубокого распараллеливания для эффективного использования аппаратуры

Существующие подходы для создания параллельных программ

- Автоматическое распараллеливание
- Библиотеки нитей
  - Win32 API
  - POSIX
- Библиотеки передачи сообщений
  - MPI
- OpenMP

Вычисление числа π



$$\int_0^1 \frac{4.0}{(1+x^2)}\,dx = \pi$$

Мы можем аппроксимировать интеграл как сумму прямоугольников:

$$\sum_{i=0}^{N} F(x_i)\Delta x \approx \pi$$

Где каждый прямоугольник имеет ширину Δx и высоту $F(x_i)$ в середине интервала

Вычисление числа π. Последовательная программа.

```
#include <stdio.h>
int main ()
{
   int n =100000, i;
   double pi, h, sum, x;
   h   = 1.0 / (double) n;
   sum = 0.0;
   for (i = 1; i <= n; i ++)
   {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
    }
    pi = h * sum;
    printf("pi is approximately %.16f”, pi);
    return 0;
}
```

Автоматическое распараллеливание

Polaris, CAPO, WPP, SUIF, VAST/Parallel, OSCAR, Intel/OpenMP, ParaWise, OPC, САПФОР

```
icc -parallel pi.c
pi.c(8): (col. 5) remark: LOOP WAS AUTO-PARALLELIZED.
pi.c(8): (col. 5) remark: LOOP WAS VECTORIZED.
pi.c(8): (col. 5) remark: LOOP WAS VECTORIZED.
```

В общем случае, автоматическое распараллеливание затруднено:

- косвенная индексация (A[B[i]]);
- указатели (ассоциация по памяти);
- сложный межпроцедурный анализ;
- циклы с зависимостью по данным, как правило не распараллеливаются.

Вычисление числа π с использованием MPI

```
#include "mpi.h"
#include <stdio.h>
int main (int argc, char *argv[])
{
   int n =100000, myid, numprocs, i;
  double mypi, pi, h, sum, x;
   MPI_Init(&argc,&argv);
   MPI_Comm_size(MPI_COMM_WORLD,&numprocs);
   MPI_Comm_rank(MPI_COMM_WORLD,&myid);
   MPI_Bcast(&n, 1, MPI_INT, 0, MPI_COMM_WORLD);
    h   = 1.0 / (double) n;
   sum = 0.0;
```

Вычисление числа π с использованием MPI

```
    for (i = myid + 1; i <= n; i += numprocs)
    {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
    }
    mypi = h * sum;
    MPI_Reduce(&mypi, &pi, 1, MPI_DOUBLE, MPI_SUM, 0, MPI_COMM_WORLD);
    if (myid == 0) printf("pi is approximately %.16f”, pi);
    MPI_Finalize();
    return 0;
}
```

Вычисление числа π с использованием Win32 API

```
#include <stdio.h>
#include <windows.h>
#define NUM_THREADS 2
CRITICAL_SECTION hCriticalSection;
double pi = 0.0;
int n =100000;
void main ()
{
  int i, threadArg[NUM_THREADS];
  DWORD threadID;
  HANDLE threadHandles[NUM_THREADS];
  for(i=0; i<NUM_THREADS; i++) threadArg[i] = i+1;
  InitializeCriticalSection(&hCriticalSection);
  for (i=0; i<NUM_THREADS; i++)  threadHandles[i] =
     CreateThread(0,0,(LPTHREAD_START_ROUTINE) Pi,&threadArg[i], 0,
    &threadID);
  WaitForMultipleObjects(NUM_THREADS, threadHandles, TRUE,INFINITE);
  printf("pi is approximately %.16f", pi);
}
```

Вычисление числа π с использованием Win32 API

```
void Pi (void *arg)
{
  int i, start;
  double h, sum, x;
  h   = 1.0 / (double) n;
  sum = 0.0;
  start = *(int *) arg;
  for (i=start; i<= n; i=i+NUM_THREADS)
  {
     x = h * ((double)i - 0.5);
     sum += (4.0 / (1.0 + x*x));
   }
   EnterCriticalSection(&hCriticalSection);
      pi += h * sum;
   LeaveCriticalSection(&hCriticalSection);
}
```

Вычисление числа π с использованием OpenMP

```
#include <stdio.h>
int main ()
{
   int n =100000, i;
   double pi, h, sum, x;
   h   = 1.0 / (double) n;
   sum = 0.0;
#pragma omp parallel for reduction(+:sum) private(x)
   for (i = 1; i <= n; i ++)
   {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
    }
    pi = h * sum;
    printf("pi is approximately %.16f”, pi);
    return 0;
}
```

Достоинства использования OpenMP вместо MPI для многоядерных процессоров

- Возможность инкрементального распараллеливания
- Упрощение программирования и эффективность на нерегулярных вычислениях, проводимых над общими данными
- Ликвидация дублирования данных в памяти, свойственного MPI-программам
- Объем памяти пропорционален быстродействию процессора. В последние годы увеличение производительности процессора достигается удвоением числа ядер, при этом частота каждого ядра снижается. Наблюдается тенденция к сокращению объема оперативной памяти, приходящейся на одно ядро. Присущая OpenMP экономия памяти становится очень важна.
- Наличие локальных и/или разделяемых ядрами КЭШей будут учитываться при оптимизации OpenMP-программ компиляторами, что не могут делать компиляторы с последовательных языков для MPI-процессов.

Достоинства использования OpenMP вместо MPI для многоядерных процессоров

Процессоры Intel® Xeon® серии 5000

X5272 2 cores	X5482 4 cores
3400 MHz	3200 MHz

Процессоры Intel® Xeon® серии 7000

7150N 2 cores	X7350 4 cores	X7460 6 cores
3500 MHz	2930 MHz	2660 MHz

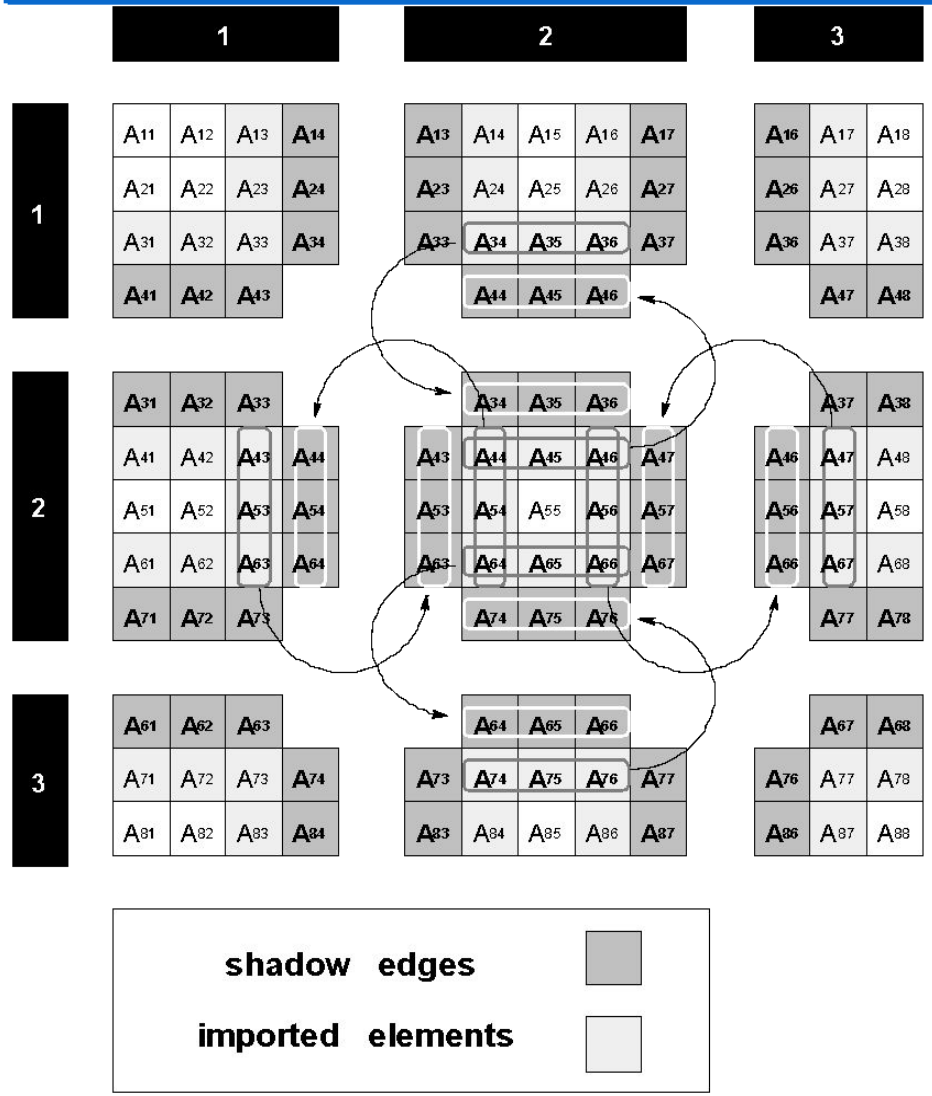
Процессоры AMD Opteron

8224 SE 2 cores	8360 SE 4 cores
3200 MHz	2500 MHz

Достоинства использования OpenMP вместо MPI для многоядерных процессоров

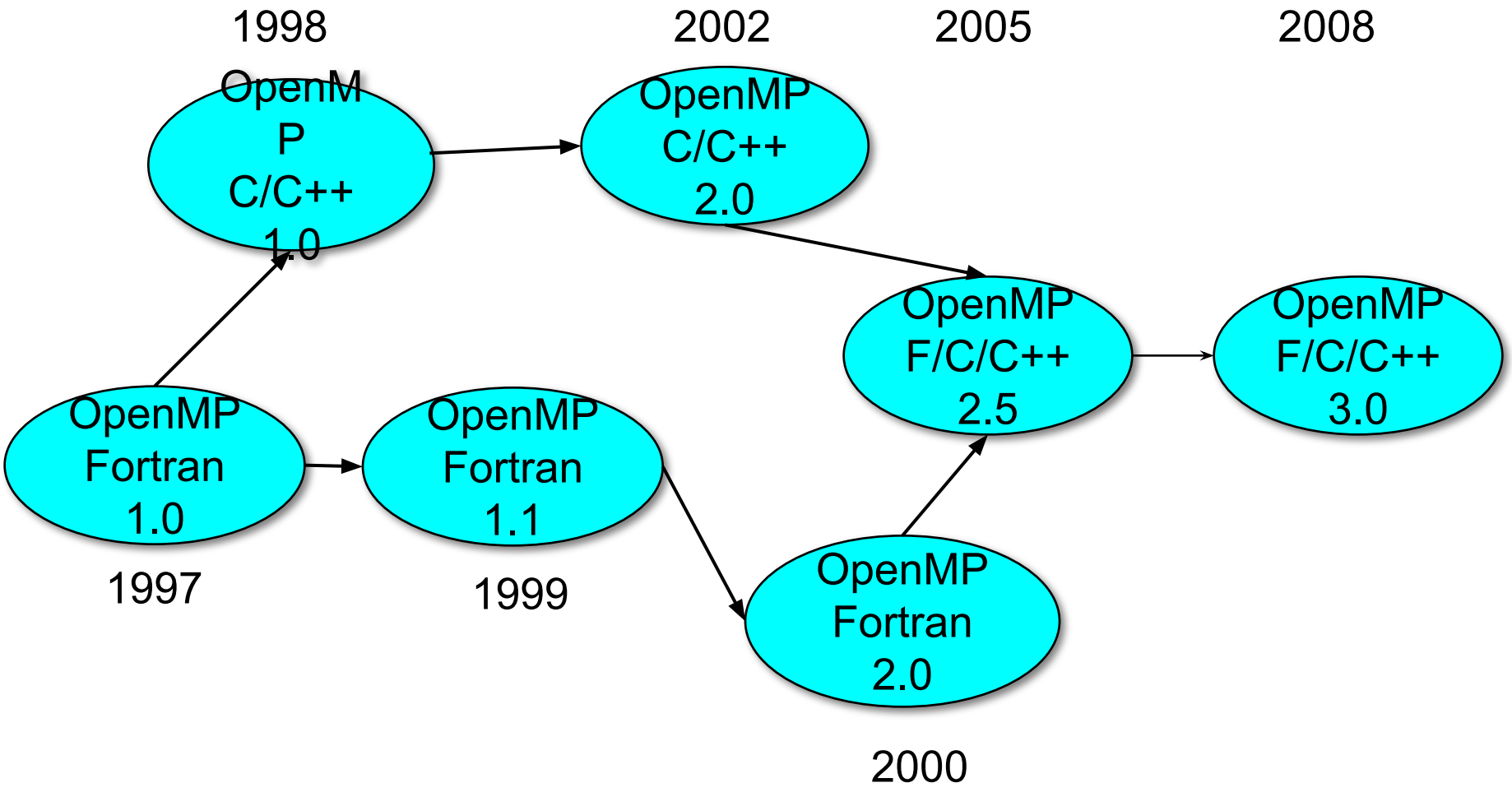
```
#define  Max(a,b) ((a)>(b)?(a):(b))
#define  L  8
#define  ITMAX 20
int i,j,it,k;
double eps, MAXEPS   = 0.5;
double A[L][L], B[L][L];
int main(int an, char **as)
{
   for (it=1; it < ITMAX; it++)  {
     eps= 0.;
     for(i=1; j<=L-2; j++)
       for(j=1; j<=L-2; j++)  {eps =  Max(fabs(B[i][j]-A[i][j]),eps);  A[i][j] = B[i][j]; }
     for(i=1; j<=L-2; j++)
       for(j=1; j<=L-2; j++)  B[i][j] = (A[i-1][j]+A[i+1][j]+A[i][j-1]+A[i][j+1])/4.;
     printf( "it=%4i   eps=%f\n", it,eps);
     if (eps < MAXEPS) break;
    }
}
```

Достоинства использования OpenMP вместо MPI для многоядерных процессоров



- Для получения программы, способной работать на кластере, необходимо распределить данные и вычисления между процессорами.
- После распределения данных требуется организовать межпроцессорные взаимодействия.
- В данном случае - для доступа к удаленным данным используются “теневые” грани, которые являются источником дублирования данных.

История OpenMP



OpenMP Architecture Review Board

- AMD
- Cray
- Fujitsu
- HP
- IBM
- Intel
- NEC
- The Portland Group, Inc.
- SGI
- Sun
- Microsoft
- ASC/LLNL
- cOMPunity
- EPCC
- NASA
- RWTH Aachen University

Компиляторы, поддеживающие OpenMP

OpenMP 3.0:

- Intel 11.0: Linux, Windows and MacOS
- Sun Studio Express 11/08: Linux and Solaris
- PGI 8.0: Linux and Windows
- IBM 10.1: Linux and AIX

Предыдущие версии OpenMP:

- GNU gcc (4.3.2)
- Absoft Pro FortranMP
- Lahey/Fujitsu Fortran 95
- PathScale
- HP
- Microsoft Visual Studio 2008 C++

Обзор основных возможностей OpenMP

```
C$OMP FLUSH
C$OMP THREADPRIVATE(/ABC/)
C$OMP PARALLEL DO SHARED(A,B,C
CALL OMP_INIT_LOCK (LCK)
C$OMP SINGLE PRIVATE(X)
SET
C$OMP PARALLEL DO ORDERED PRI
C$OMP PARALLEL  REDUCTION (-
#pragma omp parallel for private(a, b)
C$OMP BARRIER
C$OMP PARALLEL COPYIN(/blk/)
C$OMP DO LASTPRIVATE(XX)
nthrds = OMP_GET_NUM_PROCS()
omp_set_lock(lck)
```

OpenMP: API для написания многонитевых приложений

- Множество директив компилятора, набор функции библиотеки системы поддержки, переменные окружения
- Облегчает создание многонитиевых программ на Фортране, C и C++
- Обобщение опыта создания параллельных программ для SMP и DSM систем за последние 20 лет

Литература…

- http://www.openmp.org
- http://www.compunity.org
- http://www.parallel.ru/tech/tech_dev/openmp.html

Литература…

- **Гергель В.П.** Теория и практика параллельных вычислений. - М.: Интернет-Университет, БИНОМ. Лаборатория знаний, 2007.
- **Богачев К.Ю.** Основы параллельного программирования. - М.: БИНОМ. Лаборатория знаний, 2003.
- **Воеводин В.В., Воеводин Вл.В.** Параллельные вычисления. - СПб.: БХВ-Петербург, 2002.
- **Немнюгин С., Стесик О.** Параллельное программирование для многопроцессорных вычислительных систем — СПб.: БХВ-Петербург, 2002.

Литература…

Учебные курсы Интернет Университета Информационных технологий

- **Гергель В.П.** Теория и практика параллельных вычислений. — http://www.intuit.ru/department/calculate/paralltp/
- **Левин М.П.** Параллельное программирование с OpenMP http://www.intuit.ru/department/se/openmp/

Дополнительные учебные курсы:

- **Богданов А.В. и др.** Архитектуры и топологии многопроцессорных вычислительных систем. — http://www.intuit.ru/department/hardware/atmcs/
- **Барский А.Б.** Архитектура параллельных вычислительных систем. — http://www.intuit.ru/department/hardware/paralltech/
- **Барский А.Б.** Параллельное программирование. — http://www.intuit.ru/department/se/parallprog/

Вопросы?

Следующая тема

- OpenMP – модель параллелизма по управлению

Контакты

- **Бахтин В.А.**, кандидат физ.-мат. наук, заведующий сектором, Институт прикладной математики им. М.В. Келдыша РАН

  bakhtin@keldysh.ru