Суперкомпьютеры и параллельная обработка данных

Бахтин Владимир Александрович
к.ф.-м.н., ведущий научный сотрудник
Института прикладной математики им М.В.Келдыша РАН
кафедра системного программирования
факультет вычислительной математики и кибернетики
Московского университета им. М.В. Ломоносова

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);
}
```

Взаимное исключение критических интервалов

При взаимодействии через общую память нити должны синхронизовать свое выполнение.

Thread0: pi = pi + val; && Thread1: pi = pi + val;

Время	Thread 0	Thread 1
1	LOAD R1,pi	
2	LOAD R2,val	
3	ADD R1,R2	LOAD R3,pi
4	STORE R1,pi	LOAD R4,val
5		ADD R3,R4
6		STORE R3,pi

Результат зависит от порядка выполнения команд. Требуется взаимное исключение критических интервалов.

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;
}
```

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);
    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 SHMEM

```
#include <shmem.h>
#include <stdio.h>
long sync[SHMEM_REDUCE_SYNC_SIZE] = {SHMEM_SYNC_VALUE};
double work[SHMEM_REDUCE_MIN_WRKDATA_SIZE];
double pi;
int main (int argc, char *argv[])
{
    int n =100000, myid, numprocs, i;
    double h, sum, x;
    shmem_init();
    numprocs = shmem_n_pes();
    myid = shmem_my_pe();
    h   = 1.0 / (double) n;
    sum = 0.0;
```

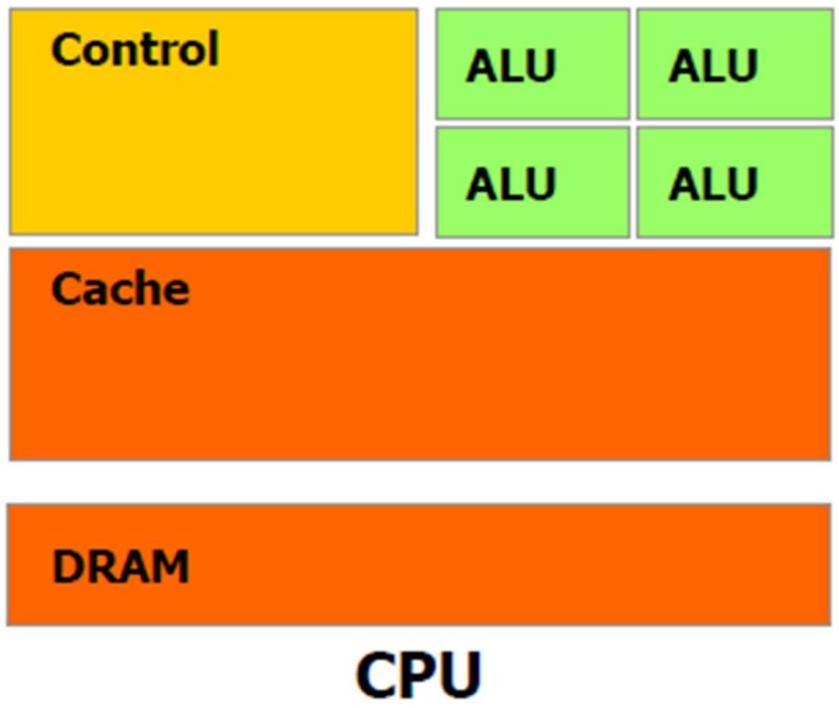
Вычисление числа π с использованием SHMEM

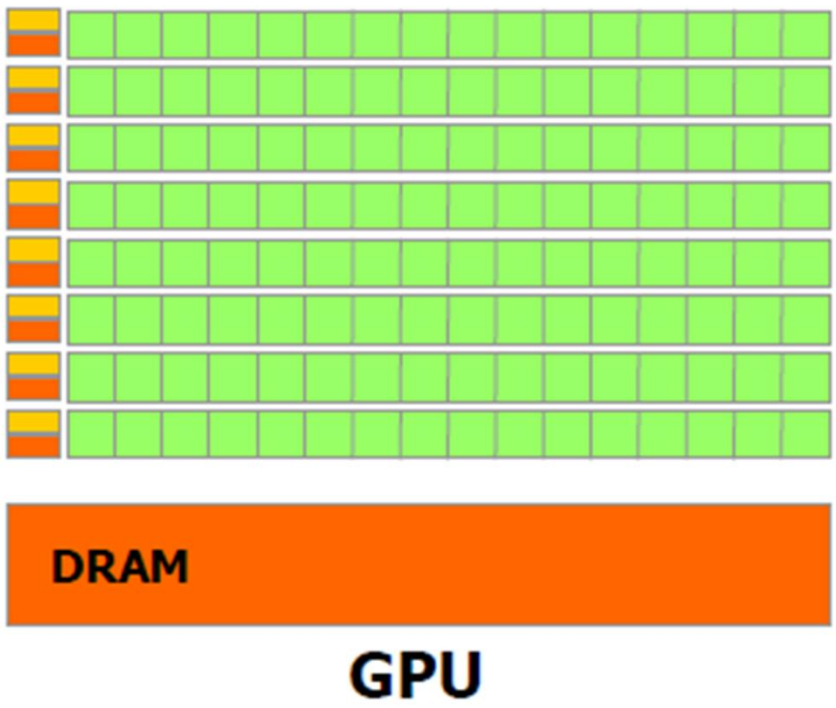
```
    for (i = myid + 1; i <= n; i += numprocs)
    {
            x = h * ((double)i - 0.5);
            sum += (4.0 / (1.0 + x*x));
    }
    pi = h * sum;
    shmem_double_sum_to_all(&pi, &pi, 1, 0, 0, numprocs, work, sync);
    if (myid == 0) printf("pi is approximately %.16f”, pi);
    shmem_finalize();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

Вычисление числа π с использованием UPC

```
#include <stdio.h>
#include <upc.h>
#include <upc_collective.h>
shared double pi;
shared double sum[THREADS];
int main ()
{
    int n =100000, i;
    double h, x;
    h   = 1.0 / (double) n;
    sum[MYTHREAD] = 0.0;
    for (i=1+MYTHREAD; i<=n;i+= THREADS)
    {
        x = h * ((double)i - 0.5);
        sum[MYTHREAD] += (4.0 / (1.0 + x*x));
     }
upc_all_reducel (&pi, sum, UPC_ADD,
  THREADS, 1,NULL,
  UPC_IN_ALLSYNC | UPC_OUT_ALLSYNC);
 if (MYTHREAD == 0)
 {
    pi *= h;
    printf("pi is approximately %.16f", pi);
 }
return 0;
}
```

CPU или GPU





Вычисление числа π с использованием CUDA

```
#include <stdio.h>
#include <cuda.h>
#define N 1000000
#define NUM_BLOCK  32  // Number of thread blocks
#define NUM_THREAD  32  // Number of threads per block
int tid;
float pi = 0;
// Kernel that executes on the CUDA device
__global__ void cal_pi(float *sum) {
      int i;
      float x, step=1.0/N; // Step size
      // Sequential thread index across the blocks
      int idx = blockIdx.x*blockDim.x+threadIdx.x;
      for (i=idx; i< N; i+=NUM_BLOCK*NUM_THREAD) {
         x = (i+0.5)*step;
         sum[idx] += 4.0/(1.0+x*x);
      }}
```

http://cacs.usc.edu/education/cs596/src/cuda/pi.cu

Вычисление числа π с использованием CUDA

```
int main(void) {  // Main routine that executes on the host
    dim3 dimGrid(NUM_BLOCK,1,1);  // Grid dimensions
    dim3 dimBlock(NUM_THREAD,1,1);  // Block dimensions
    float *sumHost, *sumDev;  // Pointer to host & device arrays
    size_t size = NUM_BLOCK*NUM_THREAD*sizeof(float);  //Array size
    sumHost = (float *)malloc(size);  // Allocate array on host
    cudaMalloc((void **) &sumDev, size); // Allocate array on device
    cudaMemset(sumDev, 0, size); // Initialize array in device to 0
    // Do calculation on device
    cal_pi <<<dimGrid, dimBlock>>> (sumDev); // call CUDA kernel
    // Retrieve result from device and store it in host array
    cudaMemcpy(sumHost, sumDev, size, cudaMemcpyDeviceToHost);
    for(tid=0; tid<NUM_THREAD*NUM_BLOCK; tid++) pi += sumHost[tid];
    pi *= step;
    printf("PI = %f\n",pi); // Print results
    free(sumHost);  // Cleanup
    cudaFree(sumDev);
    return 0;
}
```

http://cacs.usc.edu/education/cs596/src/cuda/pi.cu

Вычисление числа π с использованием OpenACC

```
#include <stdio.h>
int main ()
{
    int n =100000, i;
    double pi, h, sum, x;
    h   = 1.0 / (double) n;
    sum = 0.0;
#pragma acc parallel loop
    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;
}
```

```
pgcc -acc test.c -Minfo=all
main:
     8, Accelerator kernel generated
        Generating Tesla code
         9, #pragma acc loop gang, vector(128) /* blockIdx.x
threadIdx.x */
        12, Sum reduction generated for sum
```

Вычисление числа π с использованием DVM

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

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;
}
```

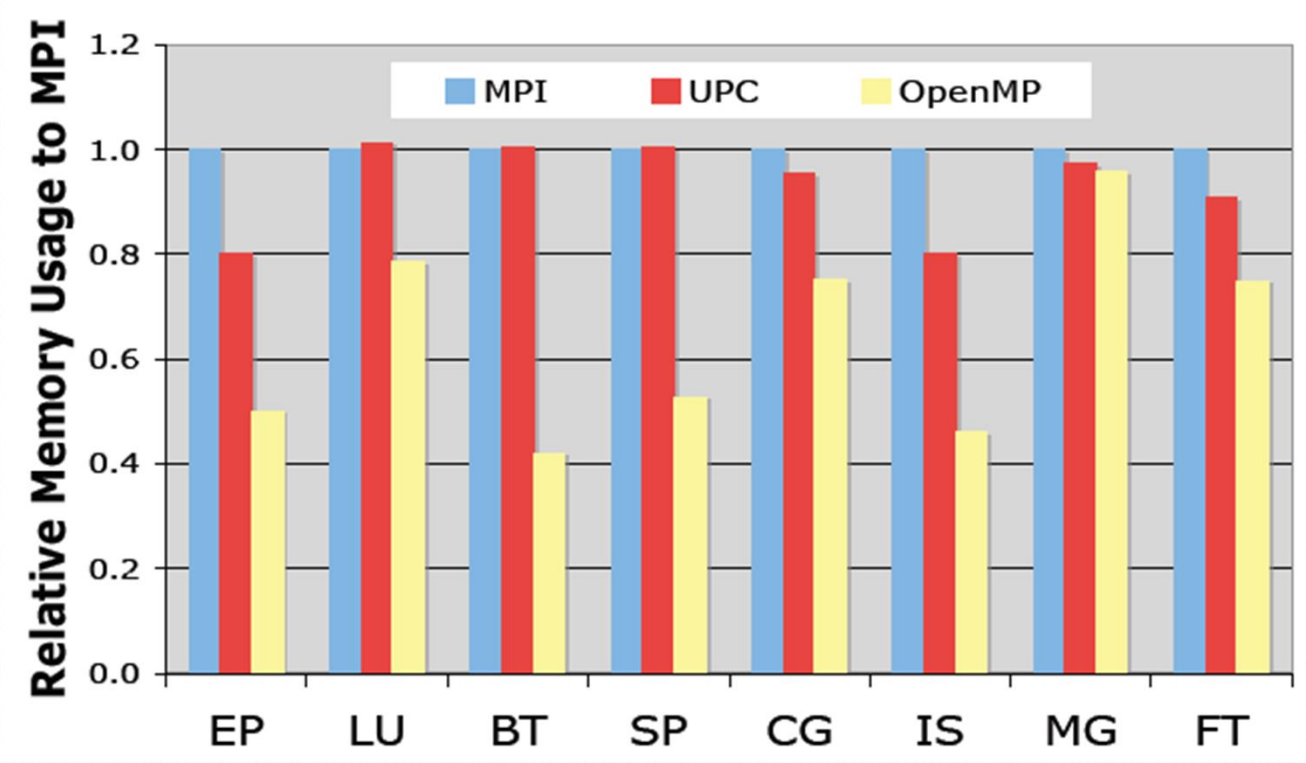
Достоинства использования OpenMP вместо MPI для многоядерных процессоров

- 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.

Тесты NAS

BT	3D Навье-Стокс, метод переменных направлений
CG	Оценка наибольшего собственного значения симметричной разреженной матрицы
EP	Генерация пар случайных чисел Гаусса
FT	Быстрое преобразование Фурье, 3D спектральный метод
IS	Параллельная сортировка
LU	3D Навье-Стокс, метод верхней релаксации
MG	3D уравнение Пуассона, метод Multigrid
SP	3D Навье-Стокс, Beam-Warning approximate factorization

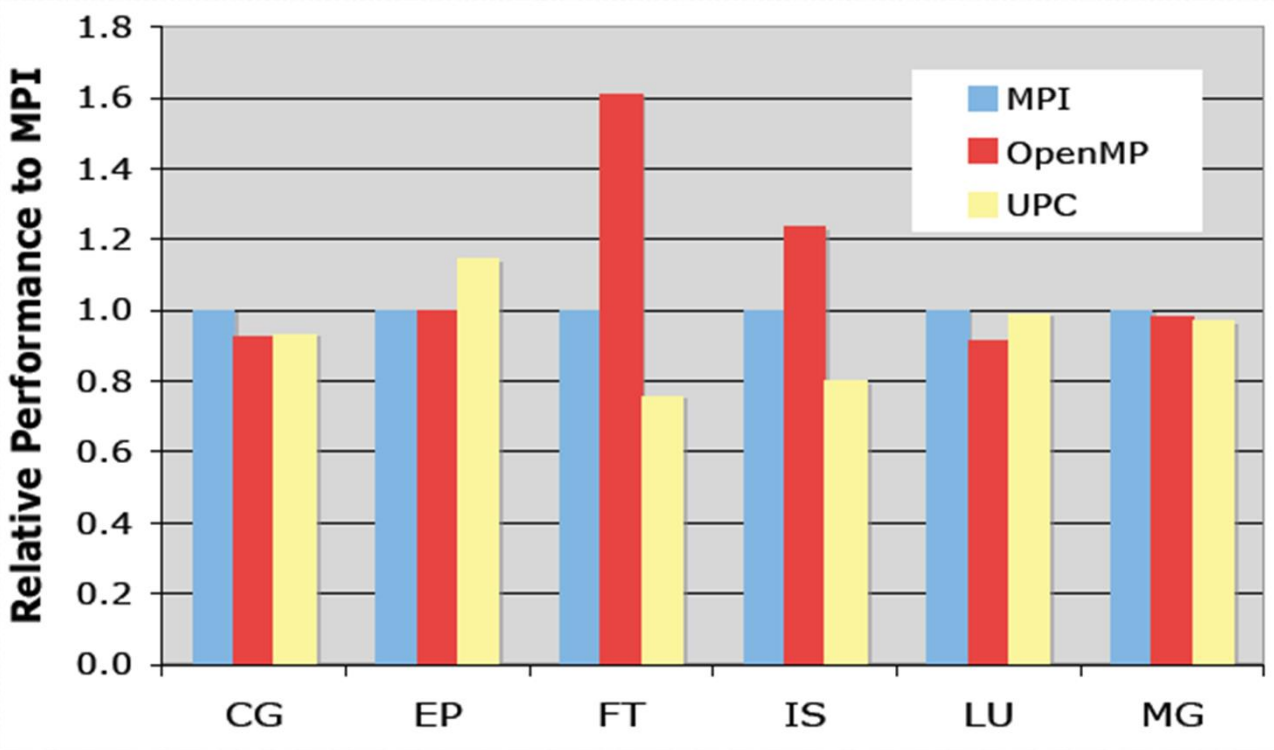
Тесты NAS



Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms

https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/Cug2010Shan.pdf

Тесты NAS

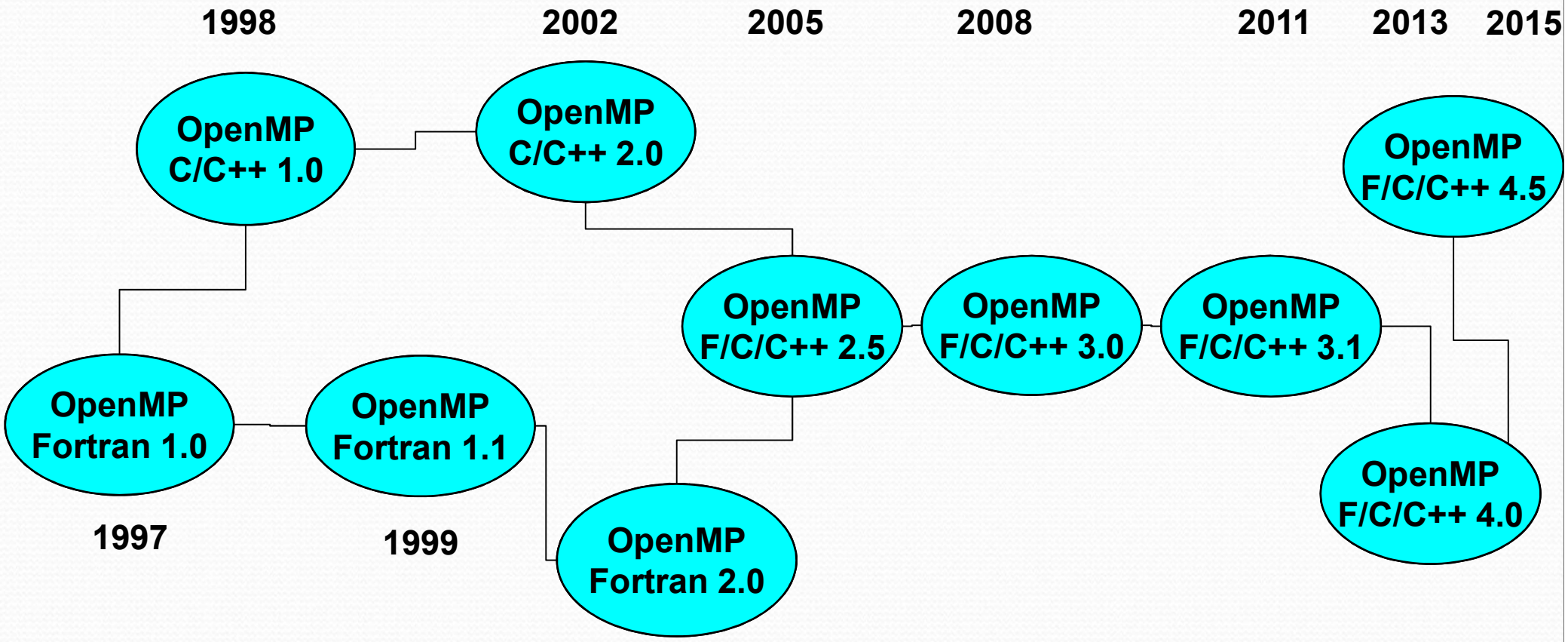


Analyzing the Effect of Different Programming Models Upon Performance and Memory Usage on Cray XT5 Platforms

https://www.nersc.gov/assets/NERSC-Staff-Publications/2010/Cug2010Shan.pdf

Содержание

История OpenMP



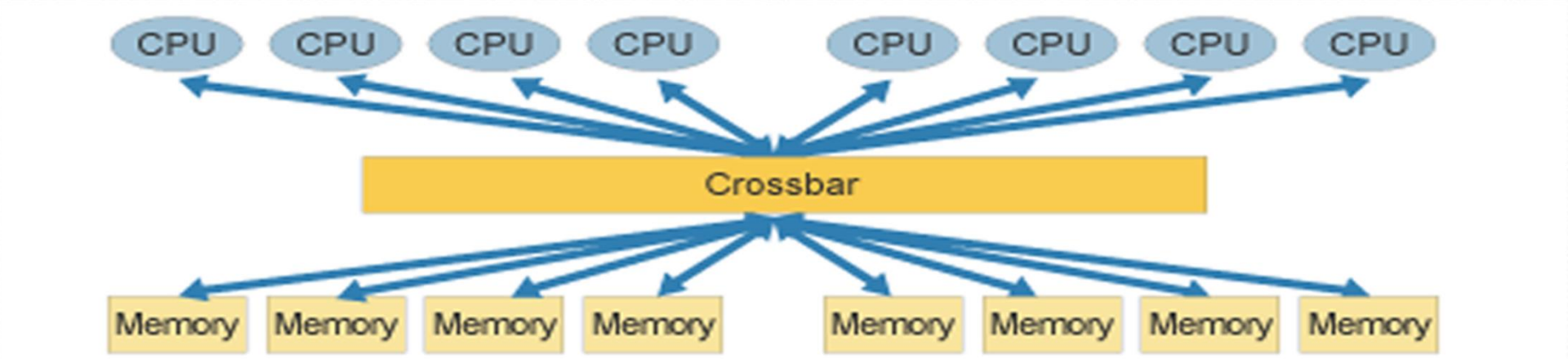
История OpenMP

Month/Year	Version
Oct 1997	Fortran 1.0
Oct 1998	C/C++ 1.0
Nov 1999	Fortran 1.1
Nov 2000	Fortran 2.0
Mar 2002	C/C++ 2.0
May 2005	OpenMP 2.5
May 2008	OpenMP 3.0
Jul 2011	OpenMP 3.1
Jul 2013	OpenMP 4.0
Nov 2018	OpenMP 5.0
Nov 2020	OpenMP 5.1
Nov 2021	OpenMP 5.2

OpenMP Architecture Review Board

- AMD
- ARM
- Cray
- Fujitsu
- HP
- IBM
- Intel
- Micron
- NEC
- NVIDIA
- Oracle Corporation
- Red Hat
- Texas Instrument
- ANL
- ASC/LLNL
- cOMPunity
- EPCC
- LANL
- LBNL
- NASA
- ORNL
- RWTH Aachen University
- Texas Advanced Computing Center
- SNL- Sandia National Lab
- BSC - Barcelona Supercomputing Center
- University of Houston

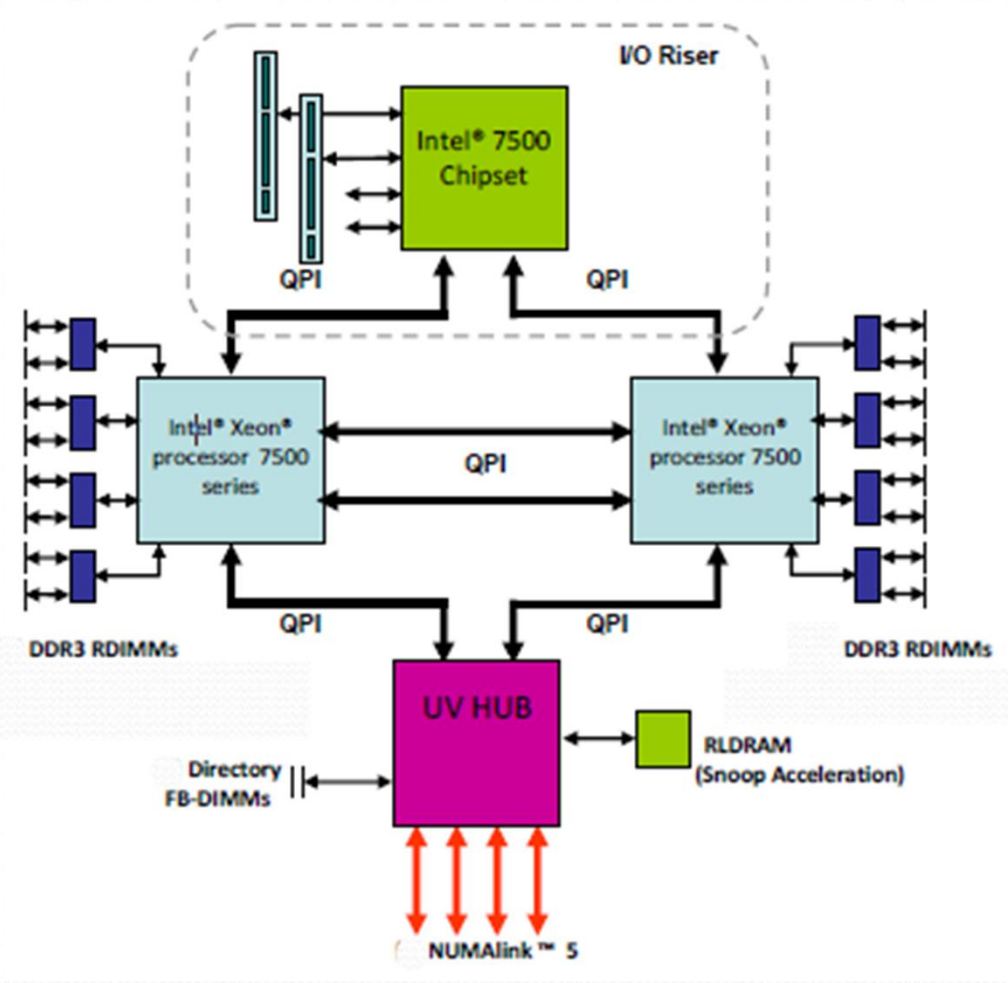
Симметричные мультипроцессорные системы (SMP)



- Все процессоры имеют доступ к любой точке памяти с одинаковой скоростью.
- Процессоры подключены к памяти либо с помощью общей шины, либо с помощью crossbar-коммутатора.
- Аппаратно поддерживается когерентность кэшей.

Например, серверы HP 9000 V-Class, Convex SPP-1200,…

Системы с неоднородным доступом к памяти (NUMA)



- Система состоит из однородных базовых модулей (плат), состоящих из небольшого числа процессоров и блока памяти.
- Модули объединены с помощью высокоскоростного коммутатора.
- Поддерживается единое адресное пространство.
- Доступ к локальной памяти в несколько раз быстрее, чем к удаленной.

Системы с неоднородным доступом к памяти (NUMA)

SGI Altix UV (UltraVioloet) 2000

- 256 Intel® Xeon® processor E5-4600 product family 2.4GHz-3.3GHz - 2048 Cores (4096 Threads)
- 64 TB памяти
- NUMAlink6 (NL6; 6.7GB/s bidirectional)

http://www.sgi.com/products/servers/uv/

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 (+: A, B)
C$OMP SECTIONS
#pragma omp parallel for private(a, b)
C$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

Директивы и клаузы

Спецификации параллелизма в OpenMP представляют собой директивы вида:

#pragma omp название-директивы*[клауза[[,]клауза]...]*

Например:

#pragma omp parallel default (none) shared (i,j)

Исполняемые директивы:

- ***barrier***
- ***taskwait***
- ***taskyield***
- ***flush***
- ***taskgroup***

Описательная директива:

- ***threadprivate***

Структурный блок

Действие остальных директив распространяется на структурный блок:

#pragma omp название-директивы*[клауза[[,]клауза]...]*

{

структурный блок

}

Структурный блок: блок кода с одной точкой входа и одной точкой выхода.

```
#pragma omp parallel
{
  …
  mainloop: res[id] = f (id);
  if (res[id] != 0) goto mainloop;
  …
  exit (0);
}
```

Структурный блок

```
#pragma omp parallel
{
  …
  mainloop: res[id] = f (id);
  …
}
if (res[id] != 0) goto mainloop;
```

Не структурный блок

Составные директивы

```
#pragma omp parallel private(i)
{
#pragma omp for firstprivate(n)
for (i = 1; i <= n; i ++)
   {
      A[i]=A[i]+ B[i];
   }
}
```

```
#pragma omp parallel for private(i) \
  firstprivate(n)
for (i = 1; i <= n; i ++)
   {
      A[i]=A[i]+ B[i];
   }
```