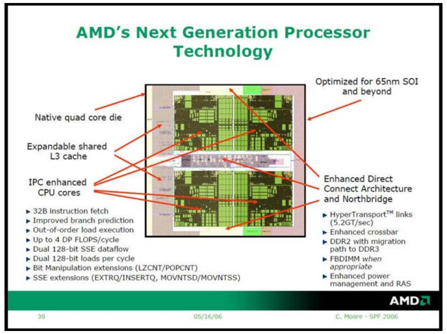
CCS HPC seminar

OpenMP tutorial

Mitsuhisa Sato
CCS, University of Tsukuba

Trends of Mulitcore processors

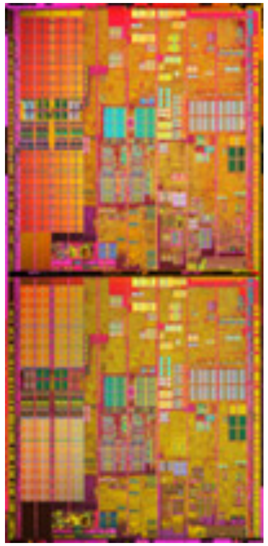
- Faster clock speed, and Finer silicon technology
  - "now clock freq is 3GHz, in future it will reach to 10GHz!?"
    - Intel changed their strategy -> multicore!
    - Clock never become faster any more
  - Silicon technology 45 nm -> 7 nm in near future!



Good news & bad news!

- Progress in Computer Architecture
  - Superpipeline, super scalar, VLIW …
  - Multi-level cache, L3 cache even in microprocessor
  - Multi-thread architecure、Intel Hyperthreading
    - Shared by multiple threads
  - Multi-core：multiple CPU core on one chip dai

Programming support is required



Inetl ® Pentium® processor Dai of Extreme-edition

Multi-core processor：Solution of Low power by parallel processing

CPU power dissipation

$$P = N \times \alpha \times C \times V^2 \times f$$

# CPU	Active rate of processors	Capacitance of circiuit	Voltage	Clock Freq
N	α	C	V	f

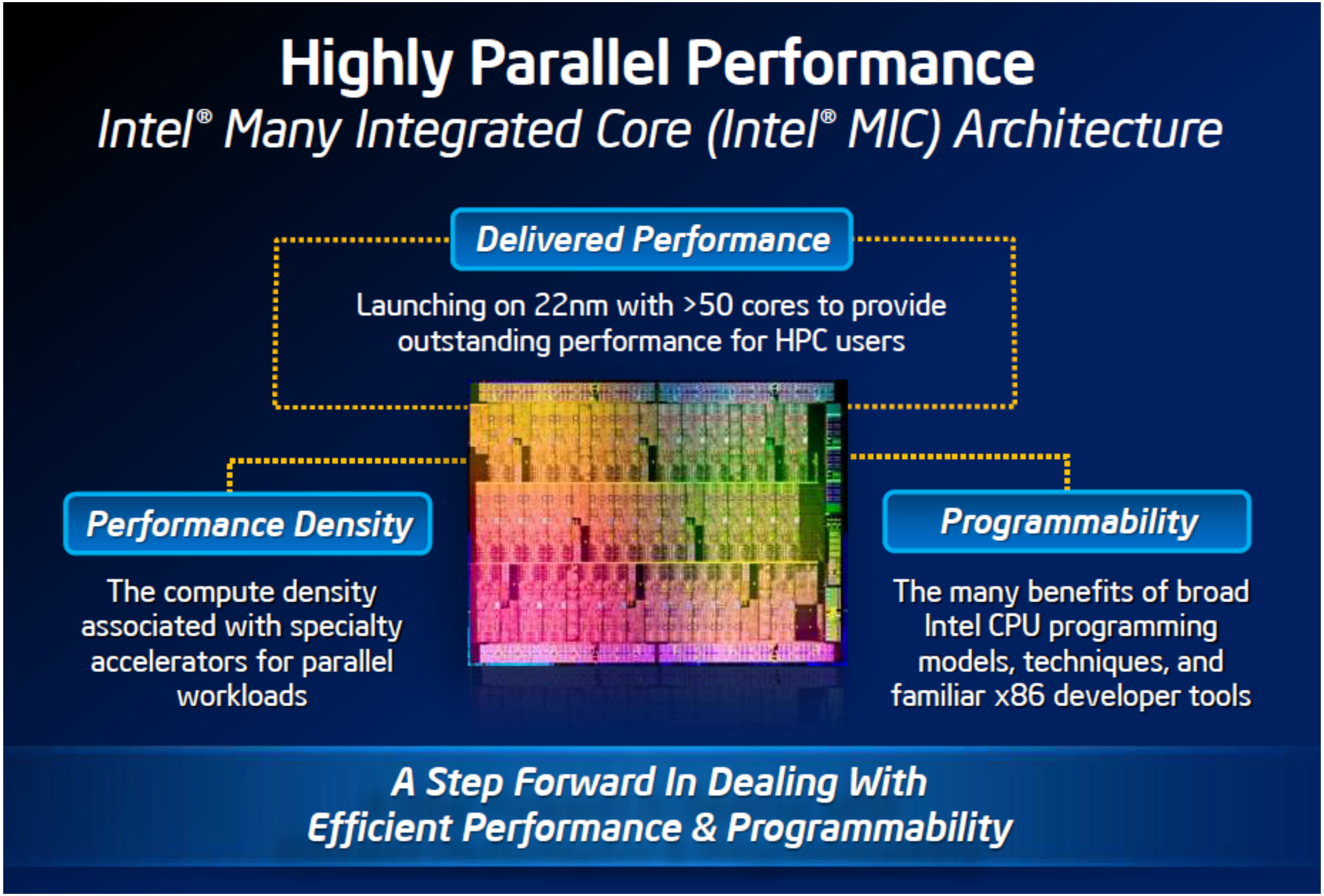
<u>Apporach for Low power by parallel processing</u>

increase N、↑ decrease V and f, ↓ improve perf. N×f ↑

- **Decreasing V and F, makes heat dissipation and power lower within a chip**
- Progress in silicon technology 130nm ⇒ 90nm⇒65nm,22nm （Decrease **C** and **V)**
- Use a silicon process for low power (embedded processor)（Small **α**）

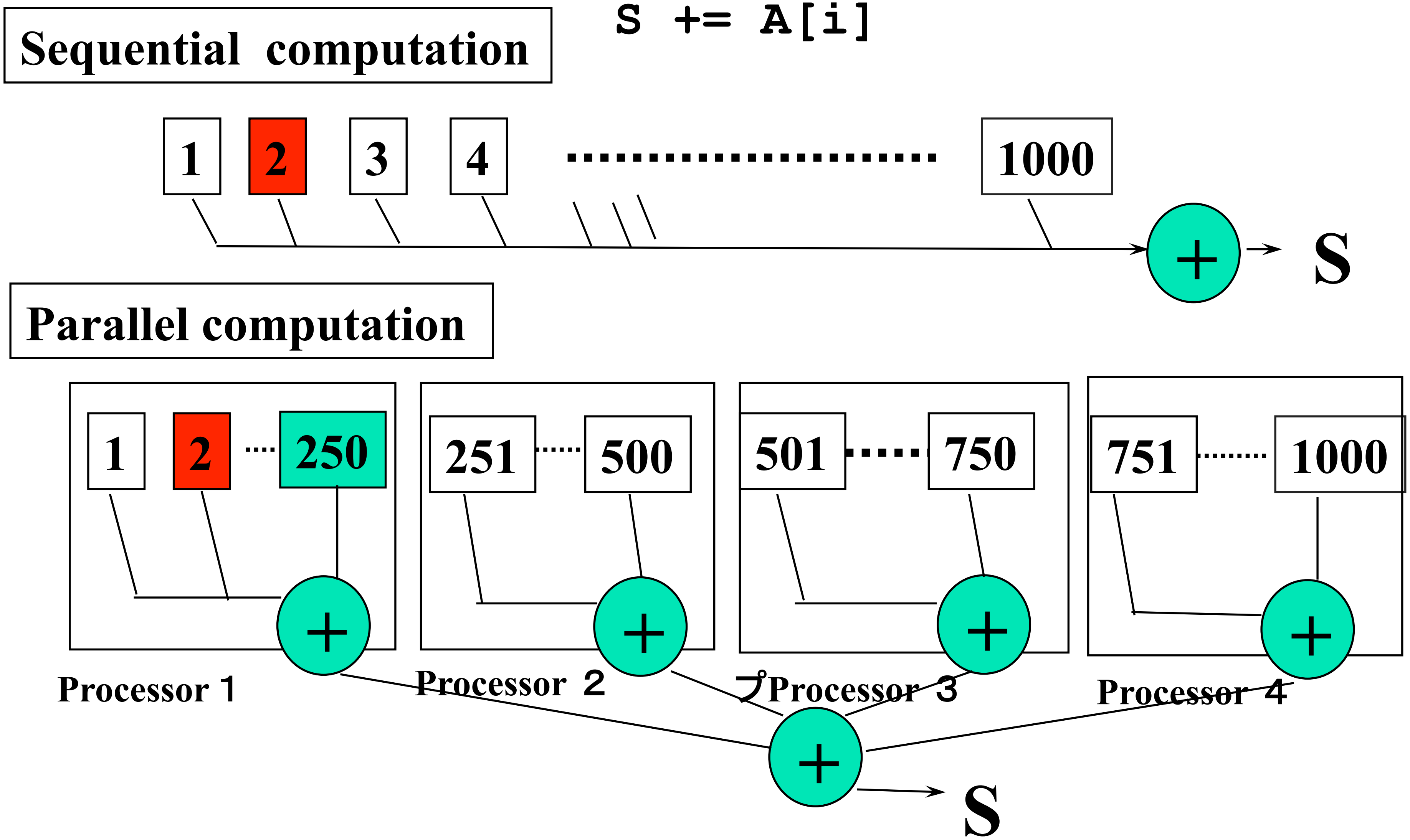
- Performance improvement by Multi-core（N=2～16)
  - Number of transistors are increasing by “Moore’s Law”
- Parallel processing by low power processor

⇒ Solution by multi-core processors for High performance embedded system


Highly Parallel Performance
Intel® Many Integrated Core (Intel® MIC) Architecture
Delivered Performance
Launching on 22nm with >50 cores to provide outstanding performance for HPC users
Performance Density
The compute density associated with specialty accelerators for parallel workloads
Programmability
The many benefits of broad Intel CPU programming models, techniques, and familiar x86 developer tools
A Step Forward In Dealing With Efficient Performance & Programmability

Very simple example of parallel computing for high performance

```
for(i=0;i<1000; i++)
    S += A[i]
```



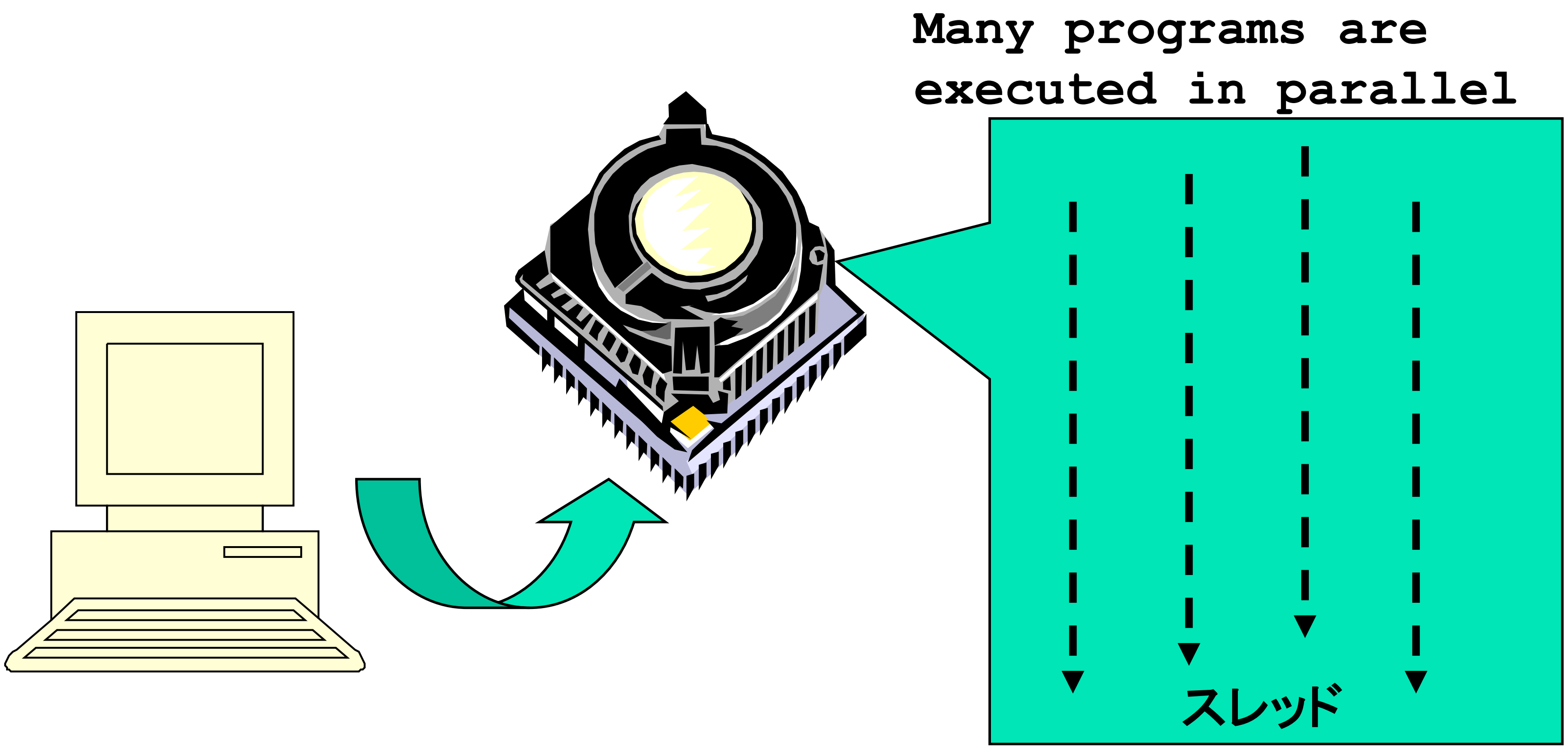
Parallel programming model

- Message passing programming model
  - Parallel programming by exchange data (message) between processors (nodes)
  - Mainly for distributed memory system (possible also for shared memory)
  - Program must control the data transfer explicitly.
  - Programming is sometimes difficult and time-consuming
  - Program may be scalable (when increasing number of Proc)

- Shared memory programming model
  - Parallel programming by accessing shared data in memory.
  - Mainly for shared memory system. (can be supported by software distributed shared memory)
  - System moves shared data between nodes (by sharing)
  - Easy to program, based on sequential version
  - Scalability is limited. Medium scale multiprocessors.

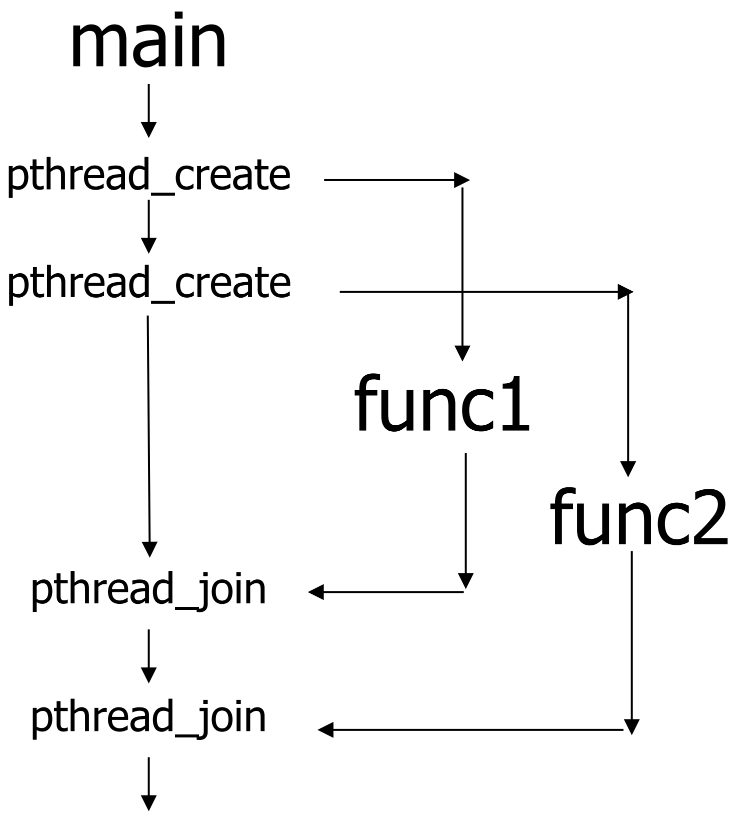
Multithread(ed) programming

- Basic model for shared memory
- Thread of execution = abstraction of execution in processors.
  - Different from process
    - Procss = thread + memory space
  - POSIX thread library = pthread



POSIX thread library

- Create thread: thread_create
- Join threads: pthread_join
- Synchronization, lock




```
#include <pthread.h>

void func1( int x );   void func2( int x );

main() {
        pthread_t t1 ;
        pthread_t t2 ;
            pthread_create( &t1, NULL,
                        (void *)func1, (void *)1 );
            pthread_create( &t2, NULL,
                        (void *)func2, (void *)2 );
            printf("main()\n");
            pthread_join( t1, NULL );
            pthread_join( t2, NULL );
}
void func1( int x ) {
      int i ;
       for( i = 0 ; i<3 ; i++ ) {
            printf("func1( %d ): %d \n",x, i );
         }
}
void func2( int x ) {
            printf("func2( %d ): %d \n",x);
}
```

Programming using POSIX thread

- Create threads

Pthread, Solaris thread

```
for(t=1;t<n_thd;t++){
  r=pthread_create(thd_main,t)
}
thd_main(0);
for(t=1; t<n_thd;t++)
      pthread_join();
```

Thread =
Execution of program

- Divide and assign iterations of loop
- Synchronization for sum

```
int s;  /* global */
int n_thd; /* number of threads */
int thd_main(int id)
{ int c,b,e,i,ss;
  c=1000/n_thd;
  b=c*id;
  e=s+c;
  ss=0;
  for(i=b; i<e; i++) ss += a[i];
  pthread_lock();
  s += ss;
  pthread_unlock();
  return s;
}
```

What's OpenMP?

- Programming model and API for shared memory parallel programming
  - It is not a brand-new language.
  - Base-languages(Fortran/C/C++) are extended for parallel programming by directives.
  - Main target area is scientific application.
  - Getting popular as a programming model for shared memory processors as multi-processor and multi-core processor appears.
- OpenMP Architecture Review Board (ARB) decides spec.
  - Initial members were from ISV compiler venders in US.
  - Oct. 1997 Fortran ver.1.0 API
  - Oct. 1998 C/C++ ver.1.0 API
  - Latest version, OpenMP 3.0
- http://www.openmp.org/

Programming using POSIX thread

- Create threads

Pthread, Solaris thread

```
for(t=1;t<n_thd;t++){
  r=pthread_create(thd_main,t)
}
thd_main(0);
for(t=1; t<n_thd;t++)
       pthread_join();
```

Thread =
Execution of program

- Divide and assign iterations of loop
- Synchronization for sum

```
int s;  /* global */
int n_thd; /* number of threads */
int thd_main(int id)
{ int c,b,e,i,ss;
  c=1000/n_thd;
  b=c*id;
  e=s+c;
  ss=0;
  for(i=b; i<e; i++) ss += a[i];
  pthread_lock();
  s += ss;
  pthread_unlock();
  return s;
}
```

Programming in OpenMP

これだけで、OK!

```
#pragma omp parallel for reduction(+:s)
   for(i=0; i<1000;i++) s+= a[i];
```

OpenMP API

- It is not a new language!
  - Base languages are extended by compiler directives/pragma, runtime library, environment variable.
  - Base languages：Fortran 90, C, C++
    - Fortran： directive line starting with !$OMP
    - C: directive by #pragma omp

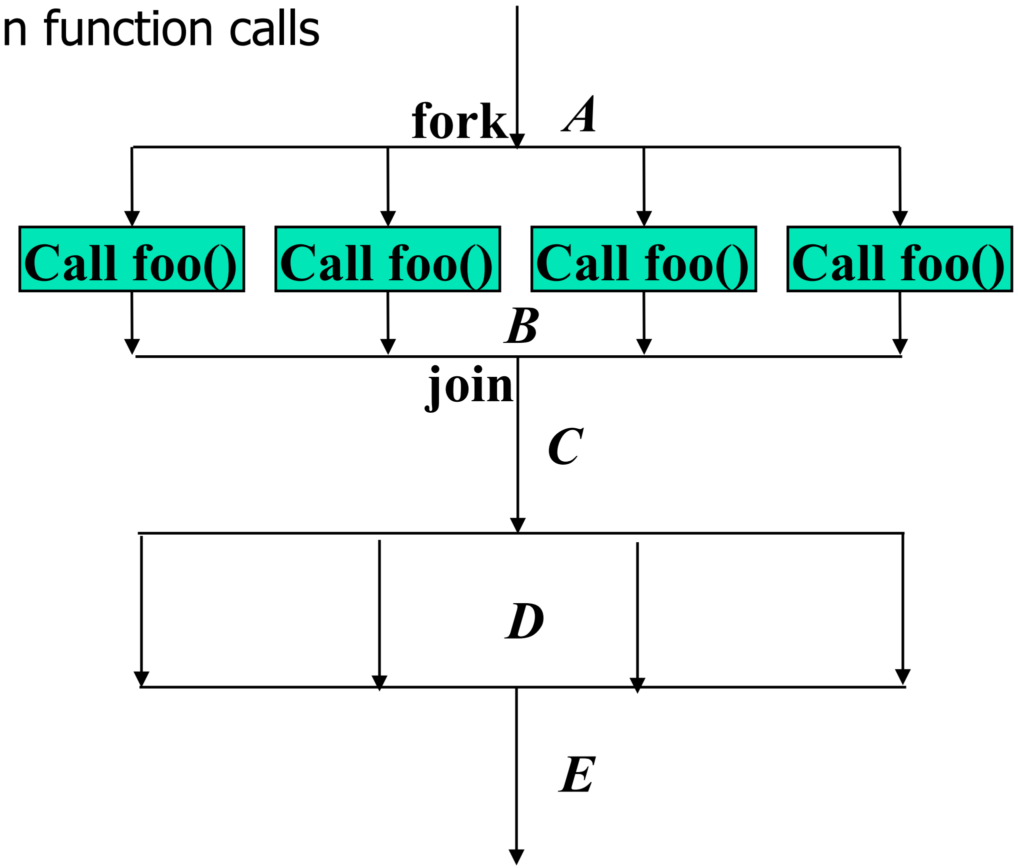
- Different from automatic parallelization
  - OpenMP parallel execution model is defined explicitly by a programmer.

- If directives are ignored (removed), the OpenMP program can be executed as a sequential program
  - Can be parallelized in incrementally
  - Practical approach with respect to program development and debugging.
  - Can be maintained as a same source program for both sequential and parallel version.

OpenMP Execution model

- Start from sequential execution
- Fork-join Model
- parallel region
  - Duplicated execution even in function calls

```
 ... A ...
#pragma omp parallel
{
    foo(); /* ..B... */
}
... C ....
#pragma omp parallel
{
... D ...
}
... E ...
```



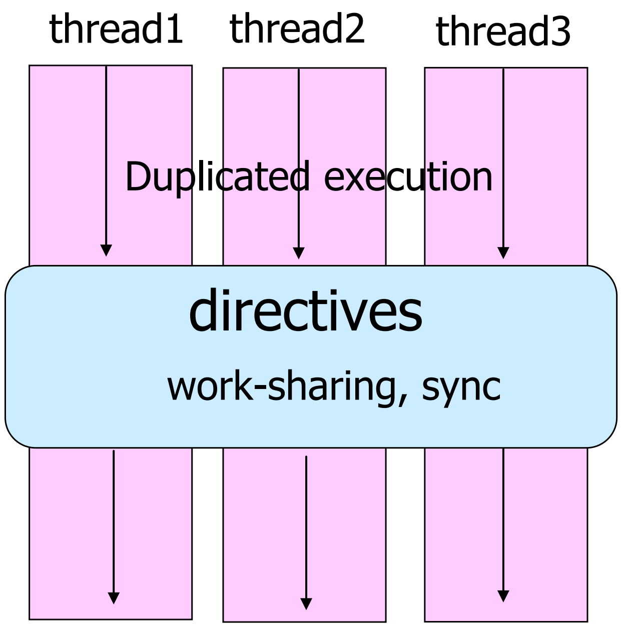
Parallel Region

- A code region executed in parallel by multiple threads (team)
  - Specified by Parallel constructs
  - A set of threads executing the same parallel region is called “team”
  - Threads in team execute the same code in region (duplicated execution)

```
#pragma omp parallel
{
    ...
    ... Parallel region...
    ...
}
```

Work sharing Constructs

- Specify how to share the execution within a team
  - Used in parallel region
  - `for` Construct
    - Assign iterations for each threads
    - For data parallel program
  - `Sections` Construct
    - Execute each section by different threads
    - For task-parallelism
  - `Single` Construct
    - Execute statements by only one thread
  - Combined Construct with parallel directive
    - `parallel for` Construct
    - `parallel sections` Construct



For Construct

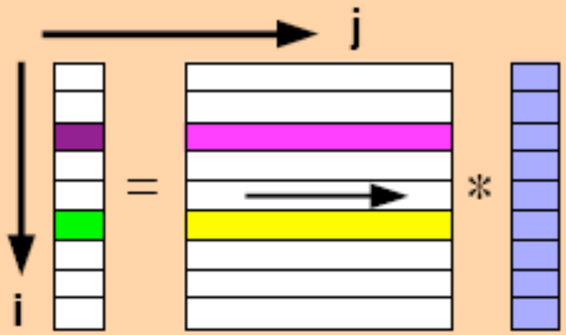
- Execute iterations specified For-loop in parallel
- For-loop specified by the directive must be in *canonical shape*

```
#pragma omp for [clause…]
 for(var=lb;  var  logical-op  ub;  incr-expr)
    body
```

  - *Var* must be loop variable of integer or pointer(automatically private)
  - *incr-expr*
    - ++*var*, *var*++, --*var*, *var*--, *var*+=*incr*, *var*-=*incr*
  - *logical-op*
    - <、<=、>、>=
  - Jump to ouside loop or break are not allows
  - Scheduling method and data attributes are specified in *clause*

Example: matrix-vector product

```
#pragma omp parallel for default(none) \
            private(i,j,sum) shared(m,n,a,b,c)
for (i=0; i<m; i++)
{
   sum = 0.0;
   for (j=0; j<n; j++)
     sum += b[i][j]*c[j];
   a[i] = sum;
 }
```



TID = 0

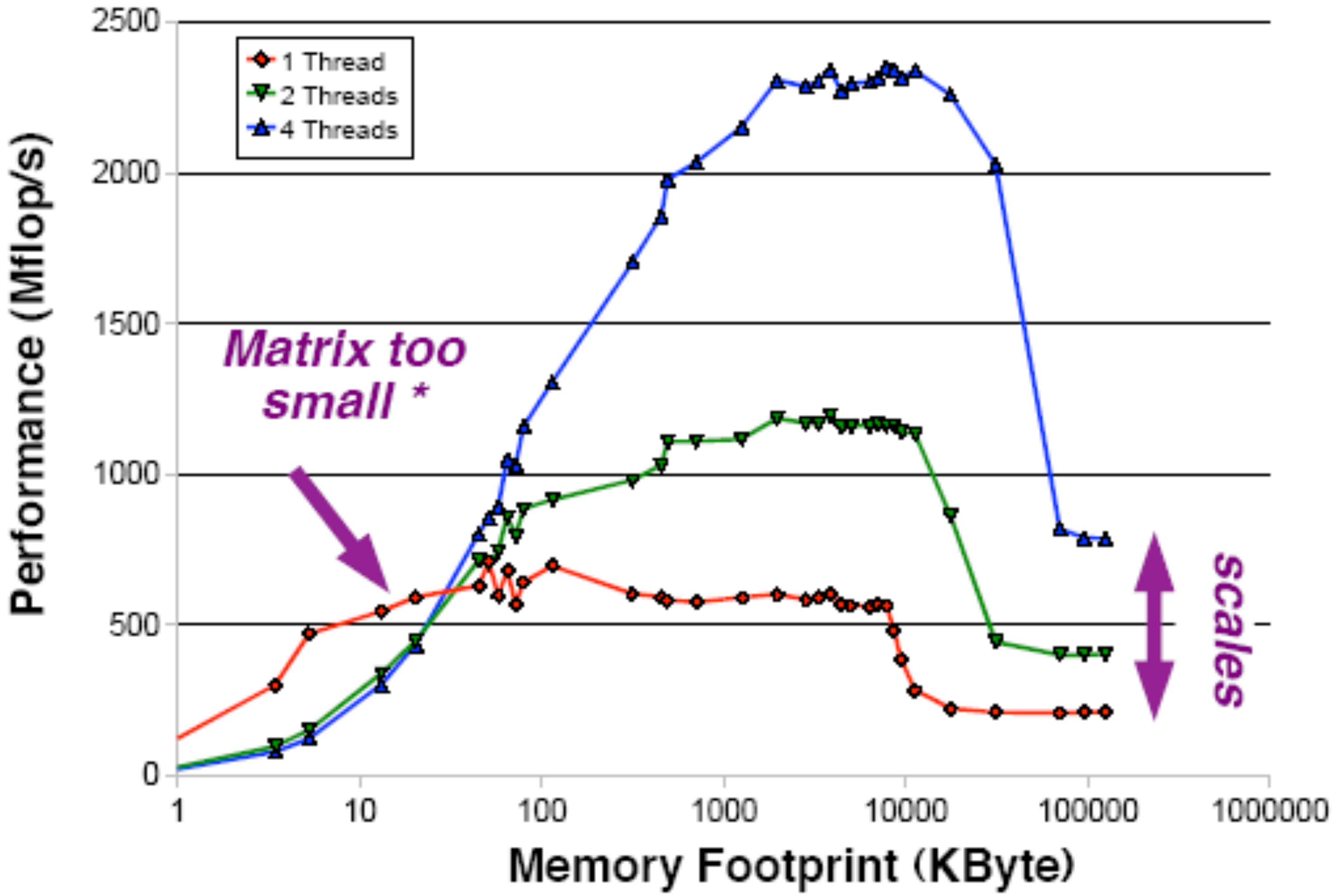
```
for (i=0,1,2,3,4)
 i = 0
  sum = Σ b[i=0][j]*c[j]
  a[0] = sum
 i = 1
  sum = Σ b[i=1][j]*c[j]
  a[1] = sum
```

TID = 1

```
for (i=5,6,7,8,9)
 i = 5
  sum = Σ b[i=5][j]*c[j]
  a[5] = sum
 i = 6
  sum = Σ b[i=6][j]*c[j]
  a[6] = sum
```

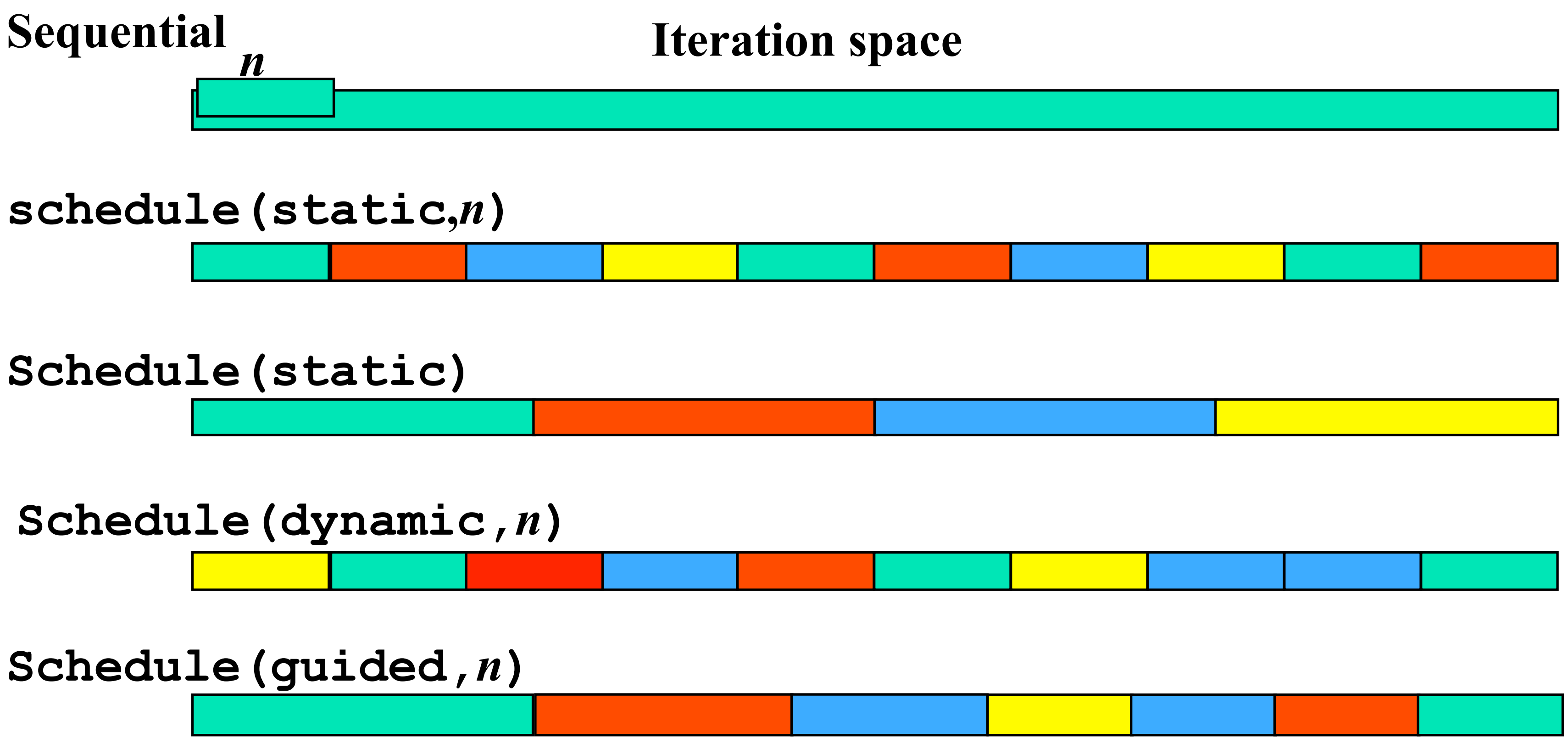
... etc ...

The performance looks like …



Scheduling methods of parallel loop

- #processor = 4



Data scope attribute clause

- Clause specified with `parallel`consruct、work sharing construct
- `shared(var_list)`
  - Specified variables are shared among threads.
- `private(var_list)`
  - Specified variables replicated as a private variable
- `firstprivate(var_list)`
  - Same as private, but initialized by value before loop.
- `lastprivate(var_list)`
  - Same as private, but the value after loop is updated by the value of the last iteration.
- `reduction(op:var_list)`
  - Specify the value of variables computed by reduction operation op.
  - Private during execution of loop, and updated at the end of loop

Barrier directive

- Sync team by barrier synchronization
  - Wait until all threads in the team reached to the barrier point.
  - Memory write operation to shared memory is completed (flush) at the barrier point.
  - Implicit barrier operation is performed at the end of parallel region, work sharing construct without `nowait` clause

```
#pragma omp barrier
```

What about performance?

- OpenMP really speedup my problem?!
- It depends on hardware and problem size/characteristics
- Esp. problem sizes is an very important factor
  - Trade off between overhead of parallelization and grain size of parallel execution.
- To understand performance, …
  - How to lock
  - How to exploit cache
  - Memory bandwidth

Advanced topics

- MPI/OpenMP Hybrid Programming
  - Programming for Multi-core cluster
- OpenMP 3.0 (2007, approved)
  - Task parallelism
- OpenACC (2012)
  - For GPU, by NVIDIA, PGI, Cray, …
- OpenMP 4.0 (2013, released)
  - Accelerator extension
  - SIMD extension
  - Task dependency description

Thread-safety of MPI

- MPI_THREAD_SINGLE
  - A process has only one thread of execution.
- MPI_THREAD_FUNNELED
  - A process may be multithreaded, but only the thread that initialized MPI can make MPI calls.
- MPI_THREAD_SERIALIZED
  - A process may be multithreaded, but only one thread at a time can make MPI calls.
- MPI_THREAD_MULTIPLE
  - A process may be multithreaded and multiple threads can call MPI functions simultaneously.
- MPI_Init_thread specifies Thread-safety level

OpenACC

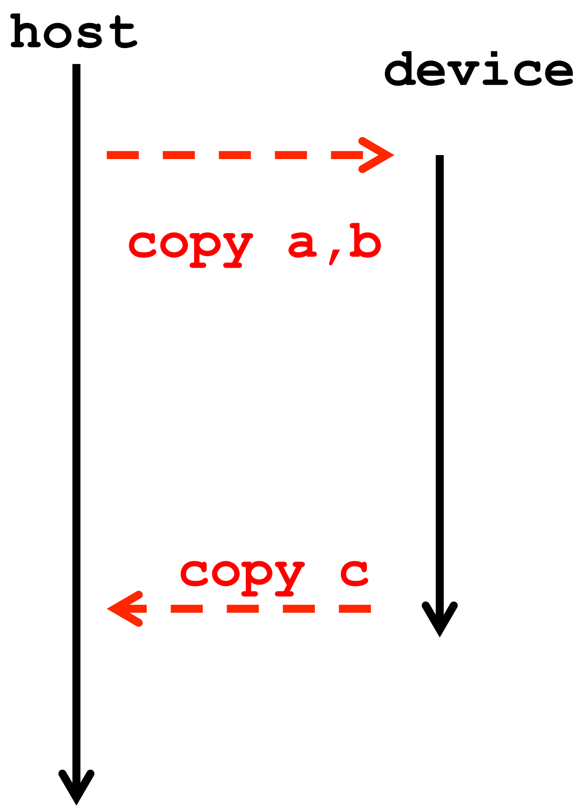
- A spin-off activity from OpenMP ARB for supporting accelerators such as GPGPU and MIC
- NVIDIA, Cray Inc., the Portland Group (PGI), and CAPS enterprise
- Directive to specify the code offloaded to GPU.

OpenACC®
DIRECTIVES FOR ACCELERATORS

A simple example

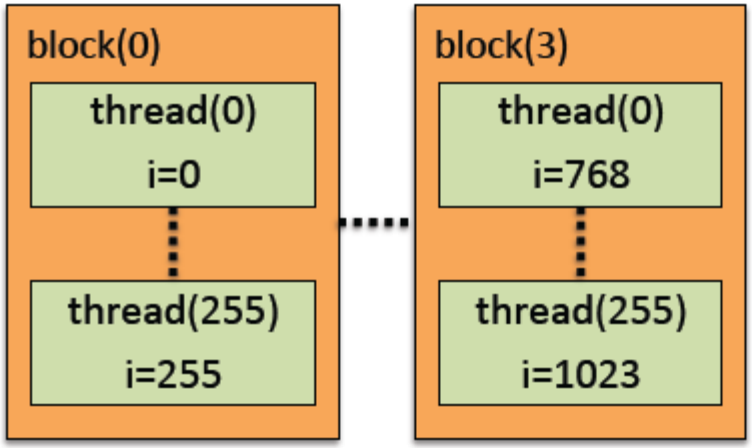
direction	copy	copyin	copyout
Host->device	○	○	
Device->Host	○		○

```
#define N 1024
int main(){
int i;
int a[N], b[N],c[N];
#pragma acc data copyin(a,b) copyout(c)
{
 #pragma acc parallel
 {
 #pragma acc loop
   for(i = 0; i < N; i++){
      c[i] = a[i] + b[i];
   }
 }
}
}
```



A simple example

```
#define N 1024
int main(){
int i;
int a[N], b[N],c[N];
#pragma acc data copyin(a,b) copyout(c)
{
 #pragma acc parallel
 {
 #pragma acc loop
   for(i = 0; i < N; i++){
      c[i] = a[i] + b[i];
   }
 }
}
}
```



execute iterations like CUDA kernel

Matrix Multiply in OpenACC

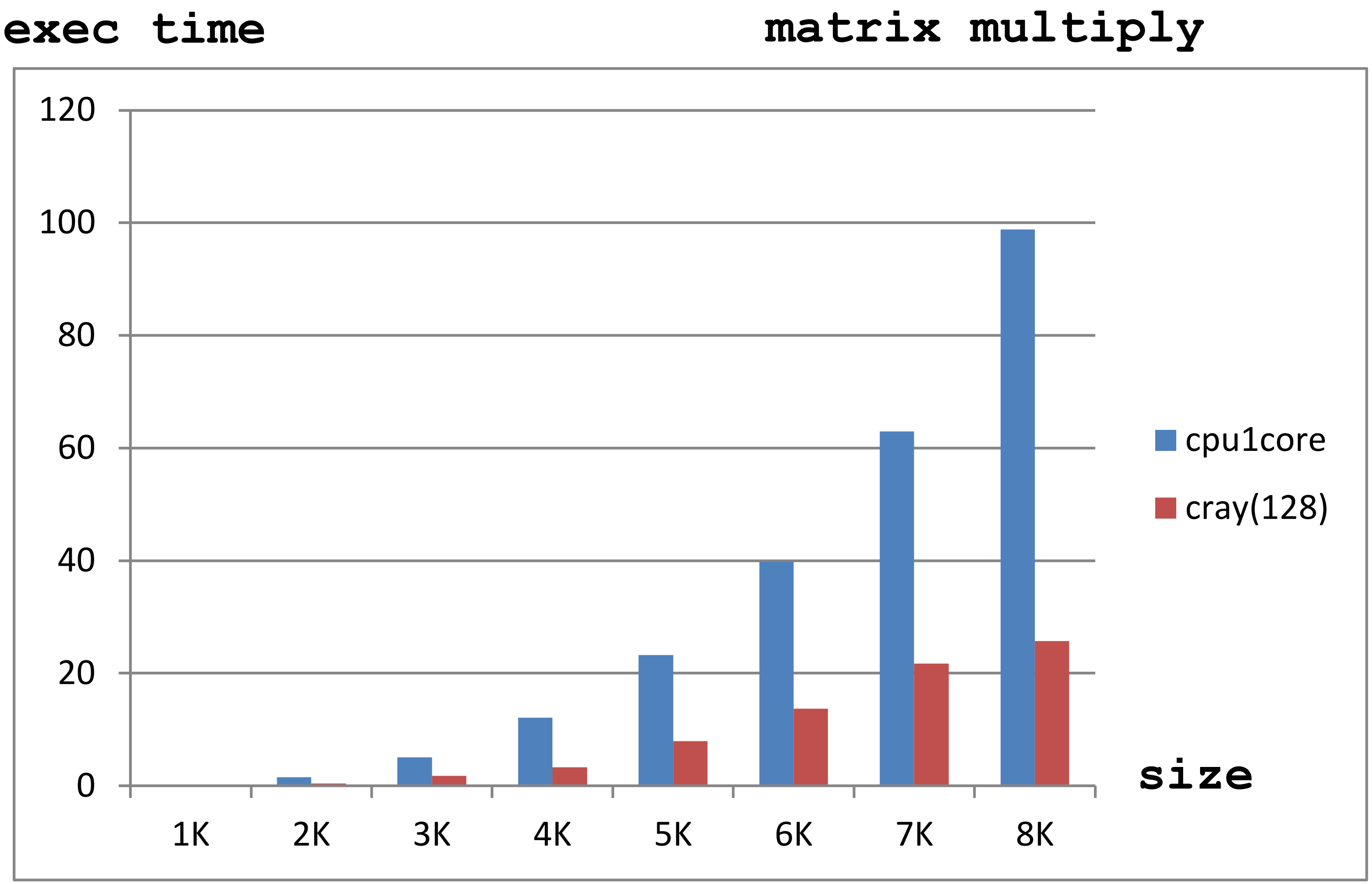
```
#define N 1024

void main(void)
{
  double a[N][N], b[N][N], c[N][N];
  int i,j;
  // ... setup data ...
#pragma acc parallel loop copyin(a, b) copyout(c)
 for(i = 0; i < N; i++){
#pragma acc loop
  for(j = 0; j < N; j++){
   int k;
   double sum = 0.0;
   for(k = 0; k < N; k++){
       sum += a[i][k] * b[k][j];
   }
   c[i][j] = sum;
  }
 }
}
```

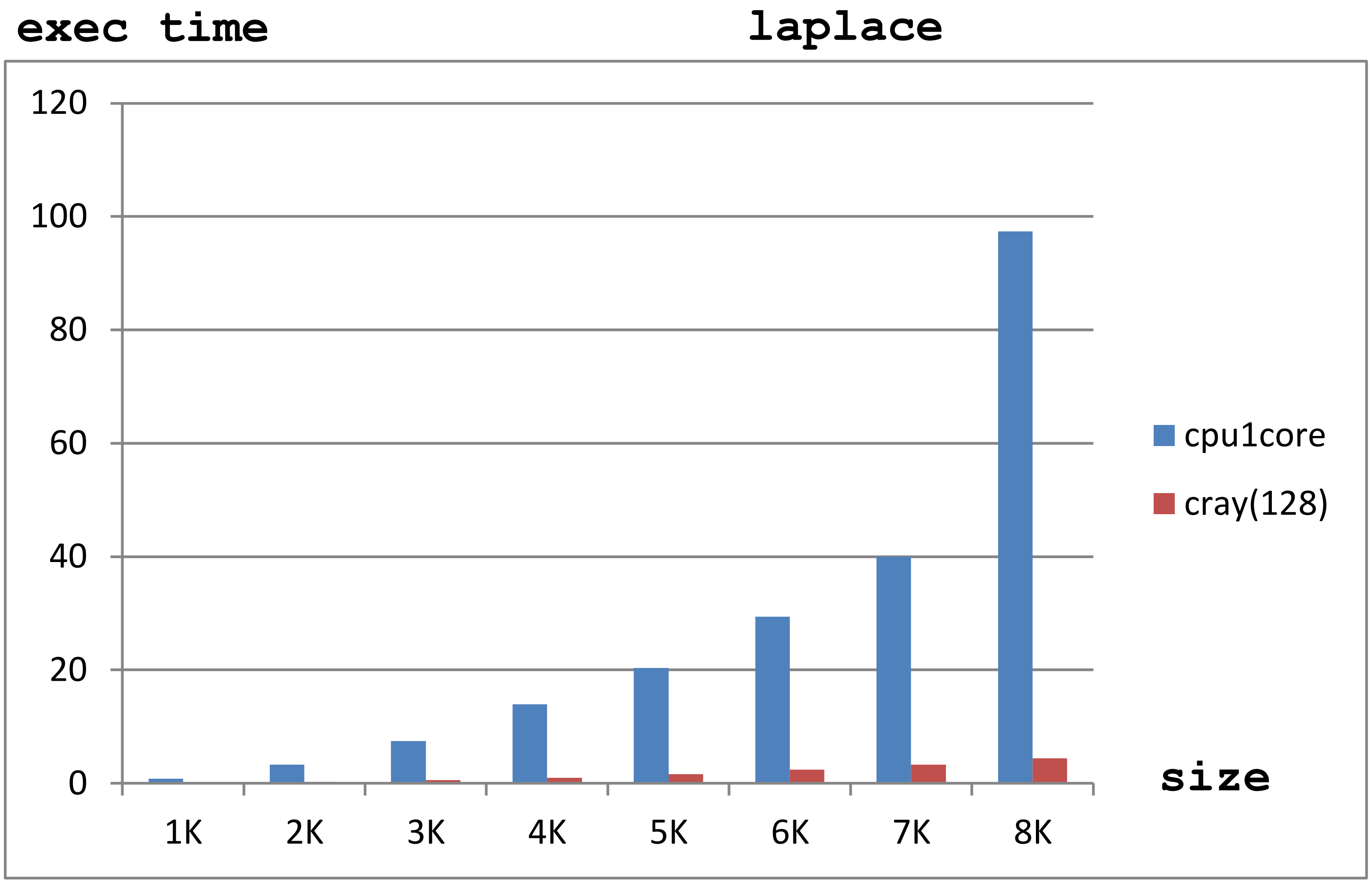
Stencil Code (Laplace Solver) in OpenACC

```
#define XSIZE 1024
#define YSIZE 1024
#define ITER 100
int main(void){
  int x, y, iter;
  double u[XSIZE][YSIZE], uu[XSIZE][YSIZE];
  // setup ...
#pragma acc data copy(u, uu)
 {
  for(iter = 0; iter < ITER; iter++){
   //old <- new
#pragma acc parallel loop
   for(x = 1; x < XSIZE-1; x++){
#pragma acc loop
         for(y = 1; y < YSIZE-1; y++)
          uu[x][y] = u[x][y];
   }
   //update
#pragma acc parallel loop
   for(x = 1; x < XSIZE-1; x++){
#pragma acc loop
         for(y = 1; y < YSIZE-1; y++)
           u[x][y] = (uu[x-1][y] + uu[x+1][y]
                    + uu[x][y-1] + uu[x][y+1]) / 4.0;
   }}
 } //acc data end
}
```

Performance of OpenACC code



Performance of OpenACC code



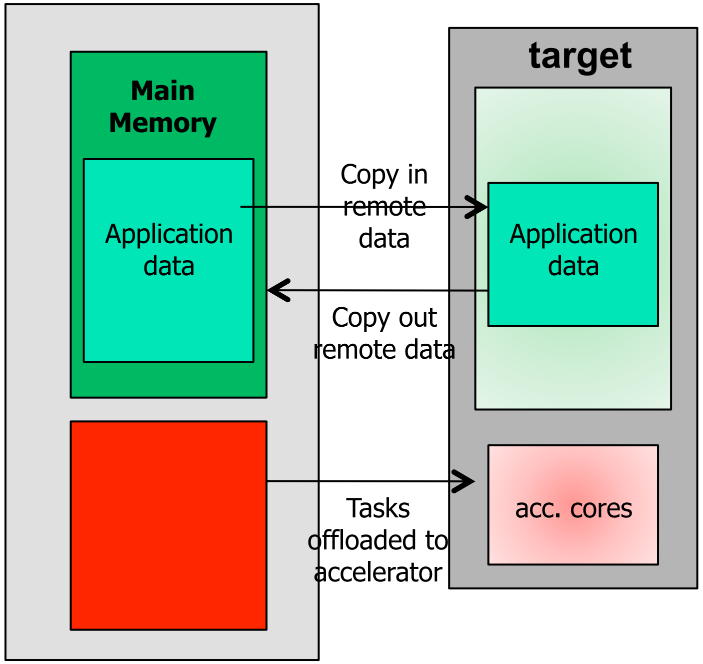
OpenMP 4.0

- Released July 2013
  - http://www.openmp.org/mp-documents/OpenMP4.0.0.pdf
  - A document of examples is expected to release soon
- Changes from 3.1 to 4.0 (Appendix E.1):
  - *Accelerator: 2.9*
  - *SIMD extensions: 2.8*
  - *Places and thread affinity: 2.5.2, 4.5*
  - *Taskgroup and dependent tasks: 2.12.5, 2.11*
  - *Error handling: 2.13*
  - *User-defined reductions: 2.15*
  - *Sequentially consistent atomics: 2.12.6*
  - *Fortran 2003 support*

slide by Yonghong@UH

Accelerator (2.9): offloading

- Execution Model: Offload data and code to accelerator
- *target* construct creates tasks to be executed by devices
- Aims to work with wide variety of accs
  - GPGPUs, MIC, DSP, FPGA, etc
  - A target could be even a remote node, intentionally




```
#pragma omp target
{
    /* it is like a new task
     * executed on a remote device */
{
```

slide by Yonghong@UH

Accelerator: explicit data mapping

- Relatively small number of truly shared memory accelerators so far
- Require the user to explicitly *map* data to and from the device memory
- Use array region

```
long a = 0x858;
long b = 0;
int anArray[100]

#pragma omp target data map(to:a) \\
  map(tofrom:b,anArray[0:64])
{
    /* a, b and anArray are mapped
      * to the device */

   /* work here */
}
/* b and anArray are mapped
  * back to the host */
```


slide by Yonghong@UH

Accelerator: hierarchical parallelism

- Organize massive number of threads
  - teams of threads, e.g. map to CUDA grid/block
- Distribute loops over teams

```
#pragma omp target

#pragma omp teams num_teams(2)
            num_threads(8)
{
     //-- creates a "league" of teams
    //-- only local barriers permitted
#pragma omp distribute
for (int i=0; i<N; i++) {

}

}
```

Only target directive makes it as accelerator region

target and map examples

```
void vec_mult(int N)
{
   int i;
   float p[N], v1[N], v2[N];
   init(v1, v2, N);
   #pragma omp target map(to: v1, v2) map(from: p)
   #pragma omp parallel for
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];
   output(p, N);
}
```

```
void vec_mult(float *p, float *v1, float *v2, int N)
{
   int i;
   init(v1, v2, N);
   #pragma omp target map(to: v1[0:N], v2[:N]) map(from: p[0:N])
   #pragma omp parallel for
   for (i=0; i<N; i++)
     p[i] = v1[i] * v2[i];
   output(p, N);
}
```

slide by Yonghong@UH

target date example

```
void vec_mult(float *p, float *v1, float *v2, int N)
{
    int i;
    init(v1, v2, N);
    #pragma omp target data map(from: p[0:N])
    {
        #pragma omp target map(to: v1[:N], v2[:N])
        #pragma omp parallel for
        for (i=0; i<N; i++)
          p[i] = v1[i] * v2[i];
        init_again(v1, v2, N);
        #pragma omp target map(to: v1[:N], v2[:N])
        #pragma omp parallel for
        for (i=0; i<N; i++)
          p[i] = p[i] + (v1[i] * v2[i]);
    }
    output(p, N);
}
```

Note mapping inheritance

slide by Yonghong@UH

teams and distribute loop example

```
float dotprod_teams(float B[], float C[], int N, int num_blocks,
  int block_threads)
{
    float sum = 0;
    int i, i0;
    #pragma omp target map(to: B[0:N], C[0:N])
    #pragma omp teams num_teams(num_blocks) thread_limit(block_threads)
      reduction(+:sum)
    #pragma omp distribute
    for (i0=0; i0<N; i0 += num_blocks)
       #pragma omp parallel for reduction(+:sum)
       for (i=i0; i< min(i0+num_blocks,N); i++)
           sum += B[i] * C[i];
    return sum;
}
```

Double-nested loops are mapped to the two levels of thread hierarchy (league and team)

slide by Yonghong@UH