Panorama des modèles et outils de programmation parallèle

Sylvain HENRY
sylvain.henry@inria.fr

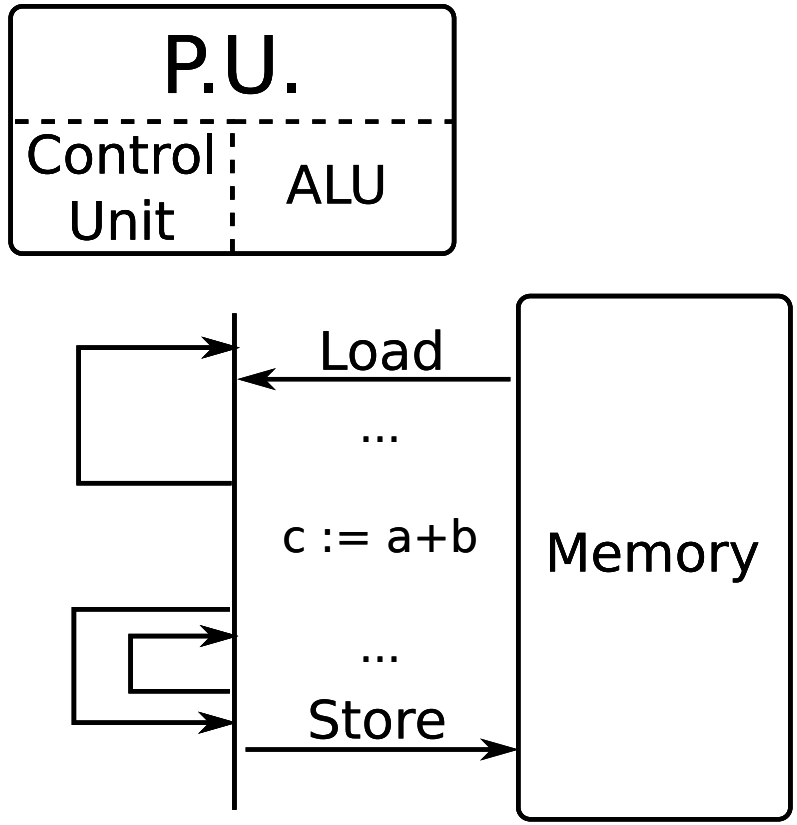
University of Bordeaux - LaBRI - Inria - ENSEIRB

April 19th, 2013

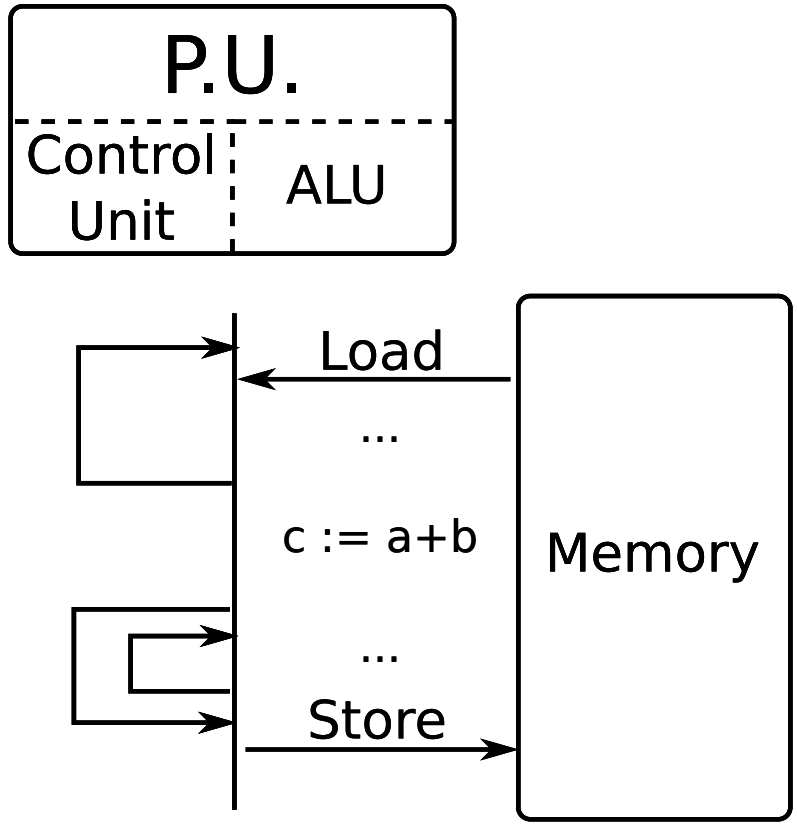
Outline

Introduction

Unicore



Unicore

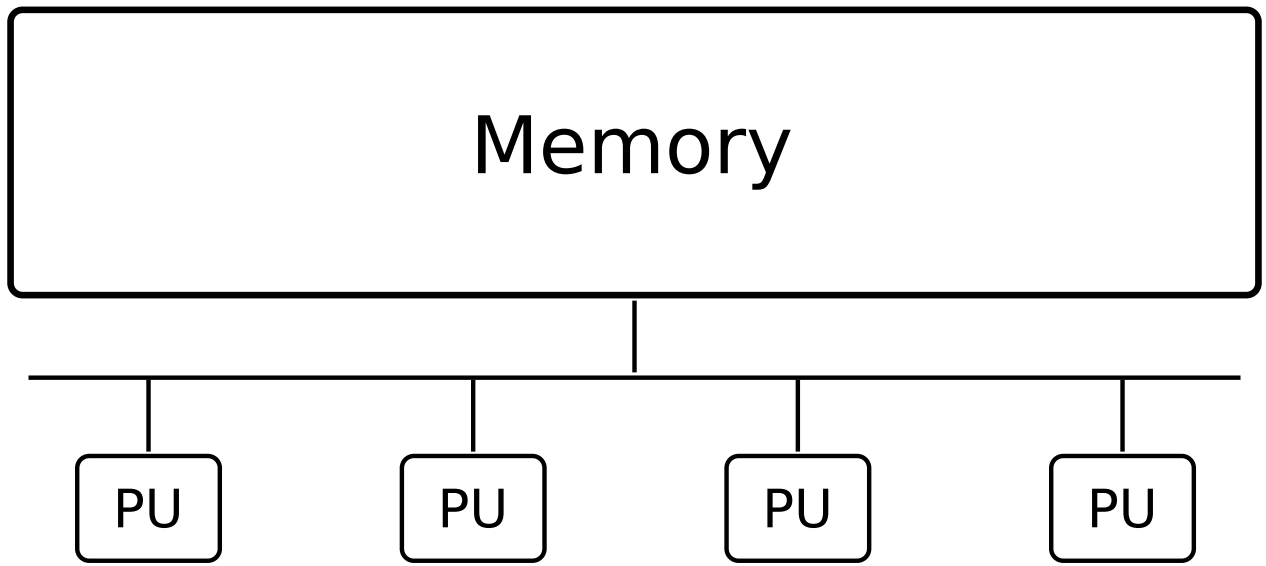


- asm (x86...), C, Fortran, C++...

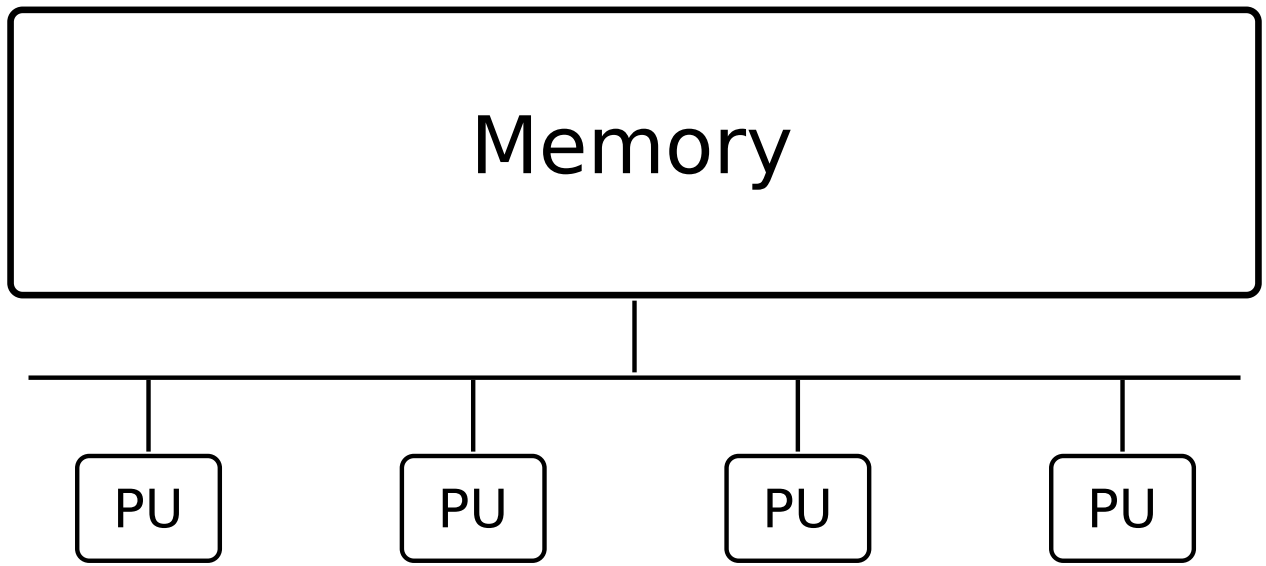
Moore's Law + Frequency Wall = Parallelism

(mandatory slide)

Multicore



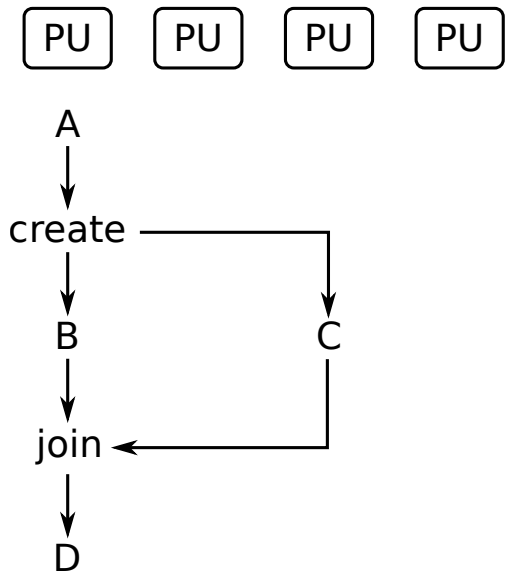
Multicore



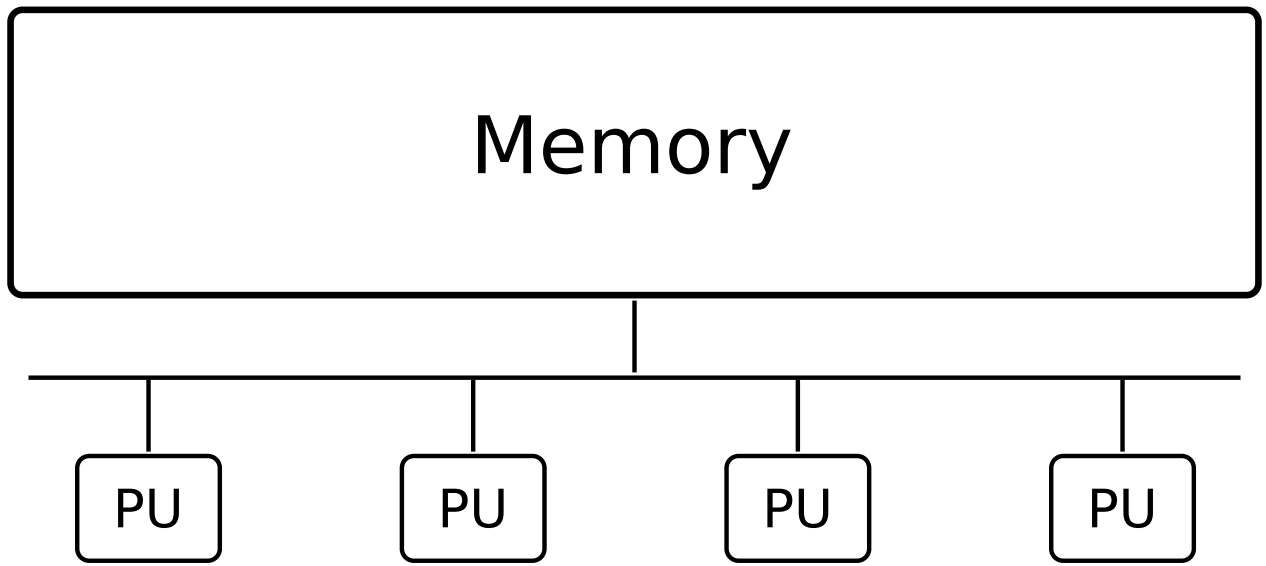
- PThread

POSIX Threads (PThread)

```c
#include <pthread.h>

void * mythread(void * arg) {
    // C
}

int main() {
    pthread_t tid;

    // A

    pthread_create(&tid, NULL, &mythread, NULL);

    // B

    pthread_join(tid, NULL);

    // D
}
```



Multicore

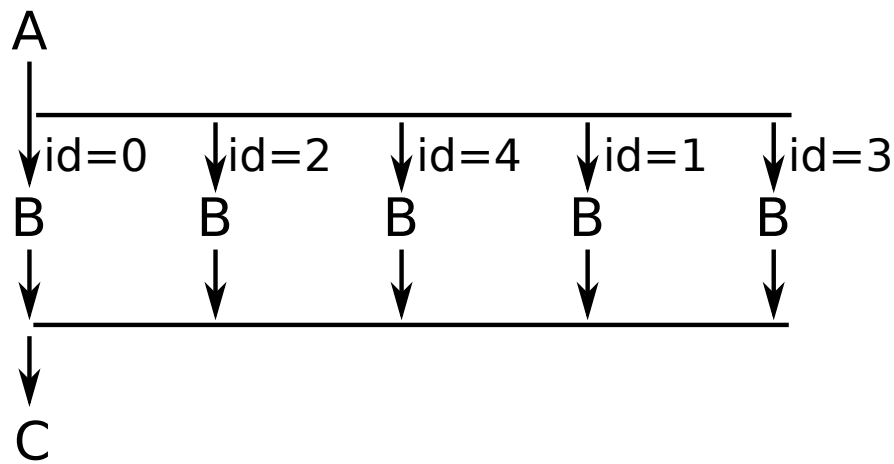


- PThread
- OpenMP

OpenMP

```c
#include <omp.h>

int main() {

    // A

    #pragma omp parallel num_threads(5)
    {
        int id = omp_get_thread_num();
        printf("My id: %d\n", id); // B
    }

    // C
}
```

```
% gcc -fopenmp test.c
% ./a.out
My id: 1
My id: 3
My id: 0
My id: 2
My id: 4
```

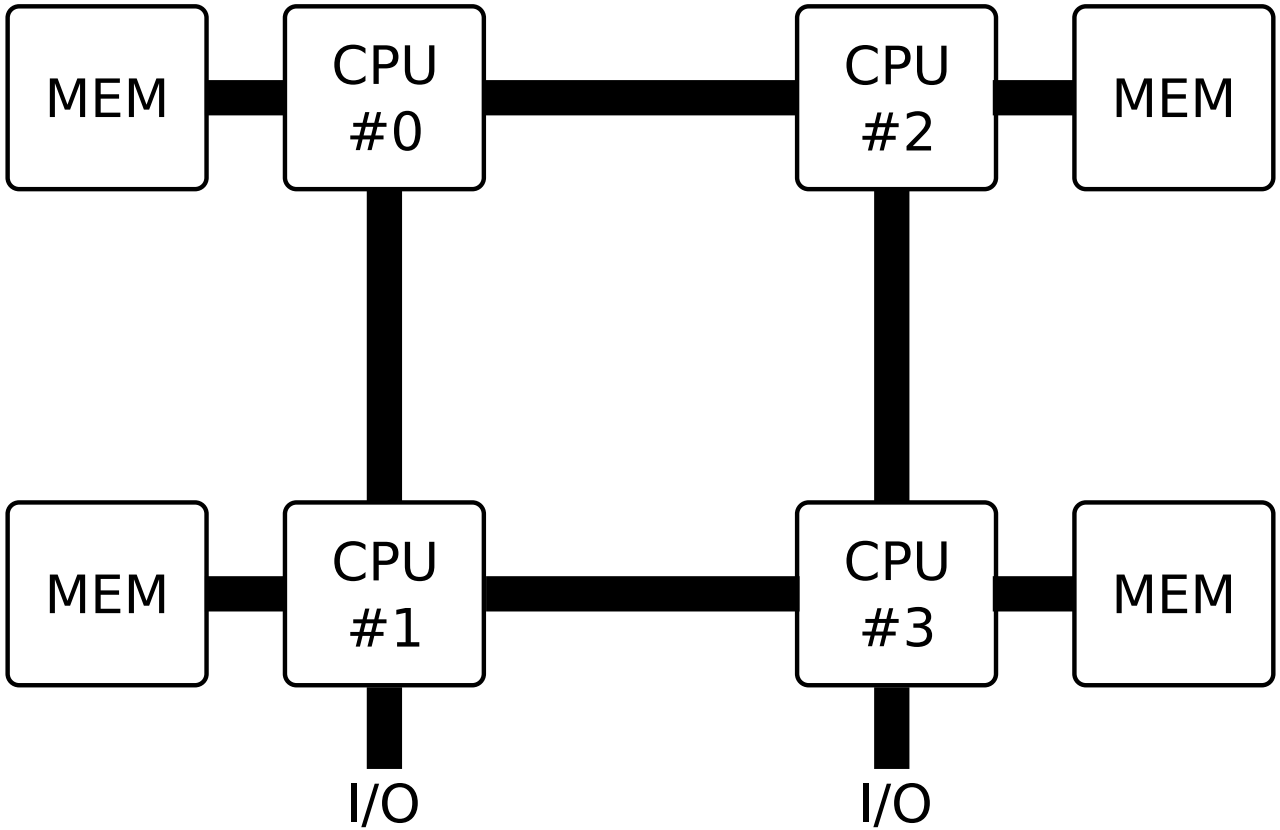


OpenMP

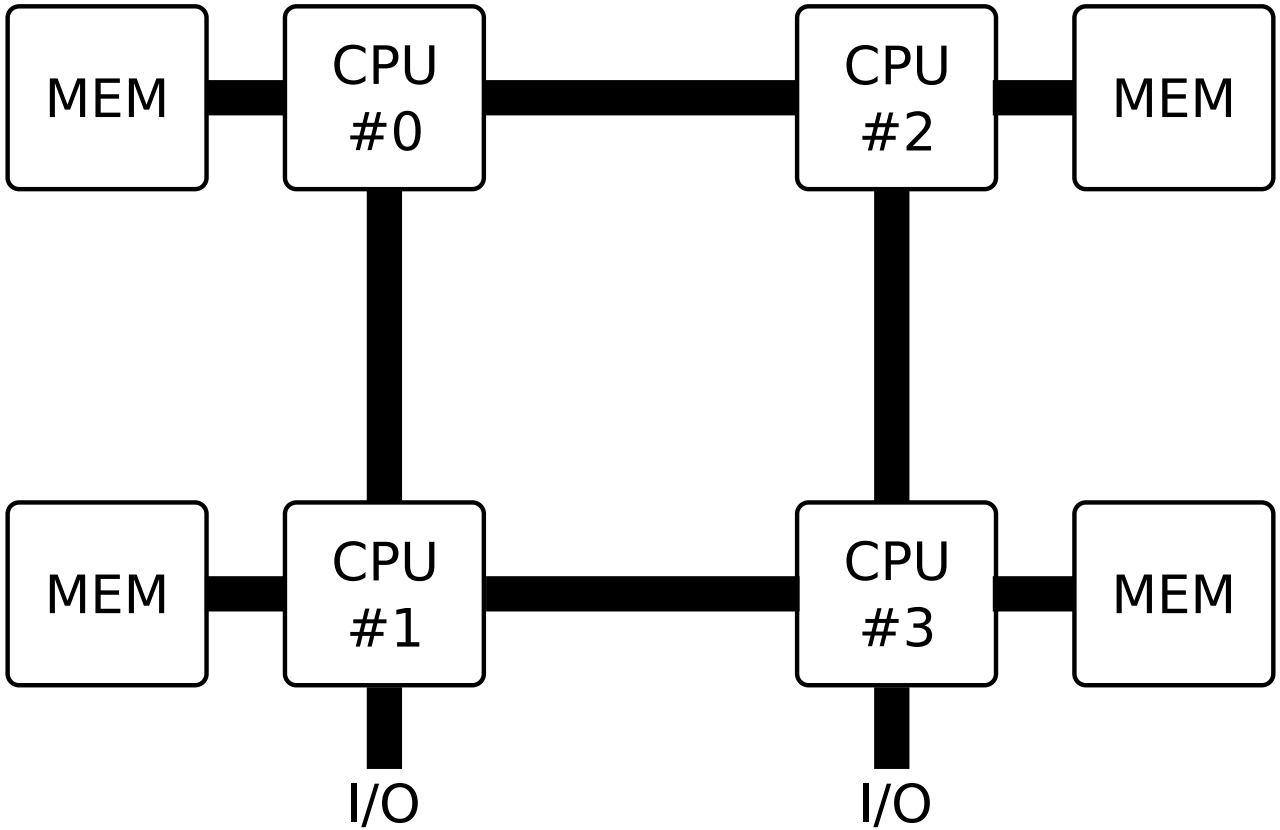
```
#pragma omp parallel for num_threads(3)
for (int i=0; i<10; i++) {
    int id = omp_get_thread_num();
    printf("%d: %d\n", id, i);
}
```

```
% ./a.out
2: 7
2: 8
2: 9
1: 4
1: 5
1: 6
0: 0
0: 1
0: 2
0: 3
```

Multicore with NUMA

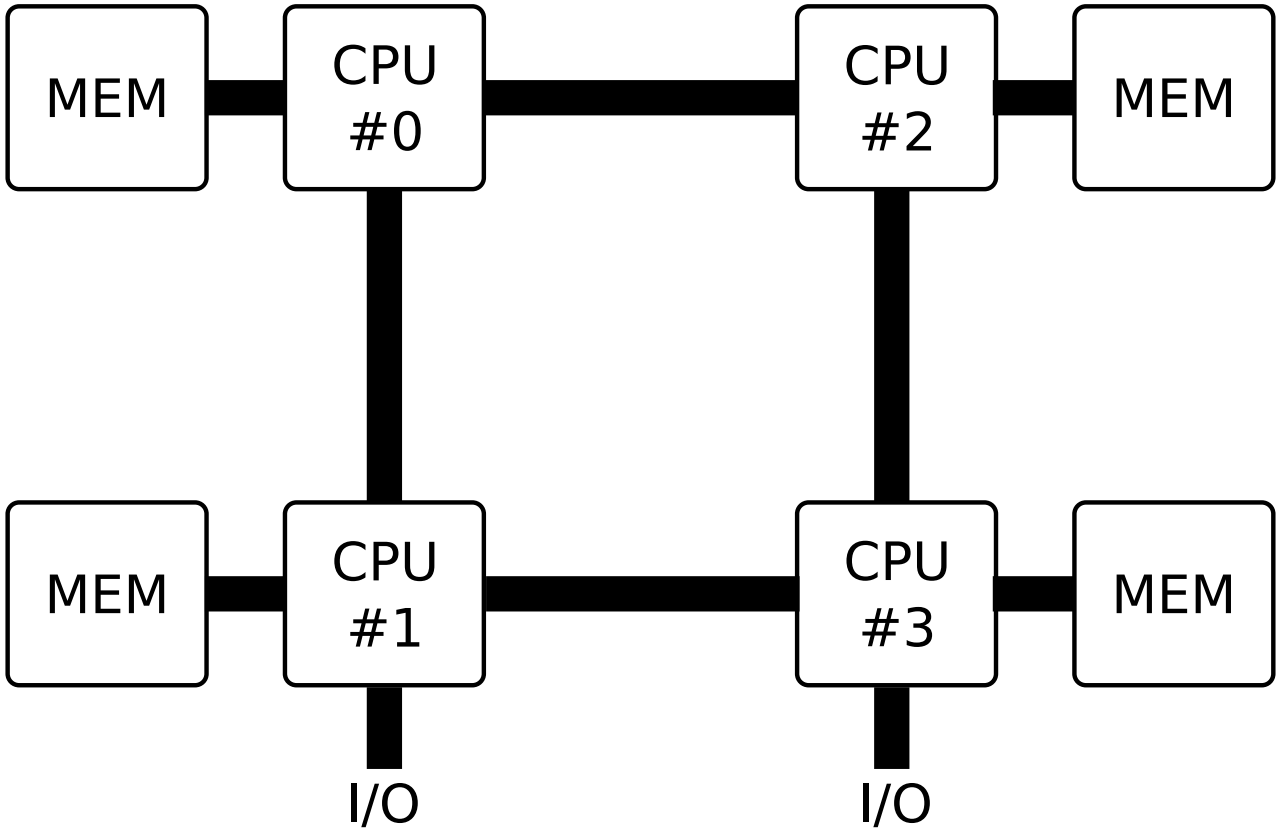


Multicore with NUMA



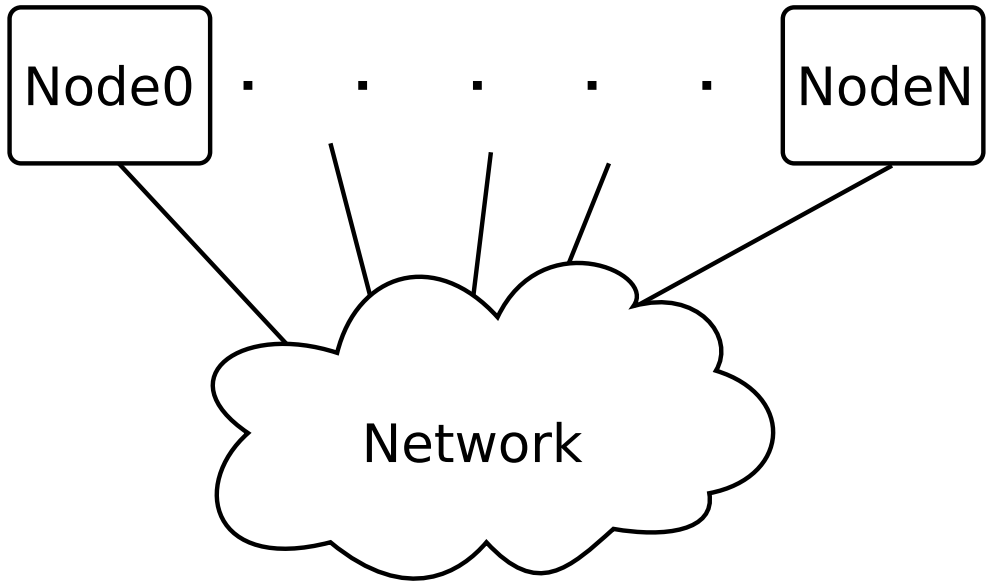
- Issues: thread scheduling (data affinity...), data migration...

Multicore with NUMA

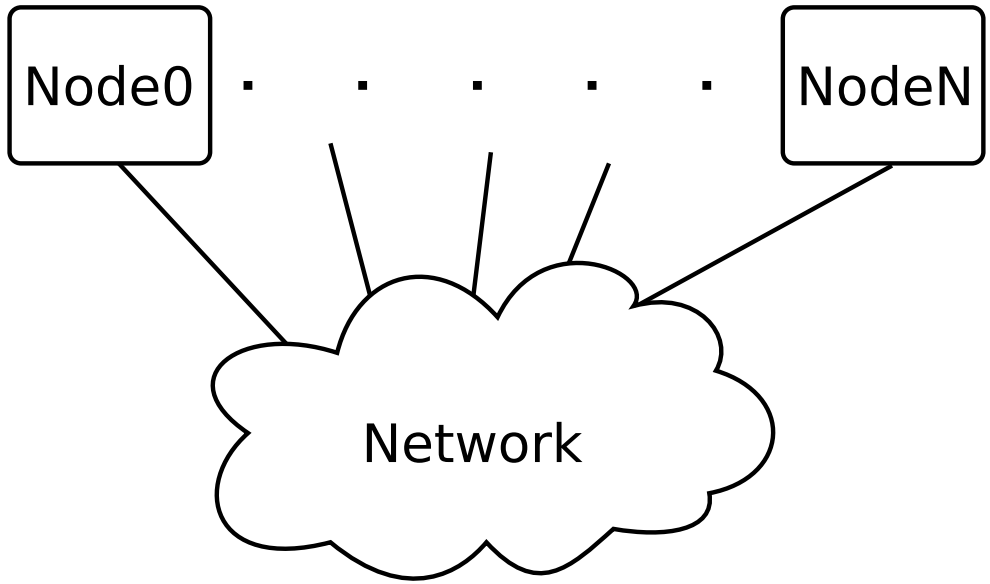


- Issues: thread scheduling (data affinity...), data migration...
- e.g. OpenMP: ForestGomp

Clusters



Clusters

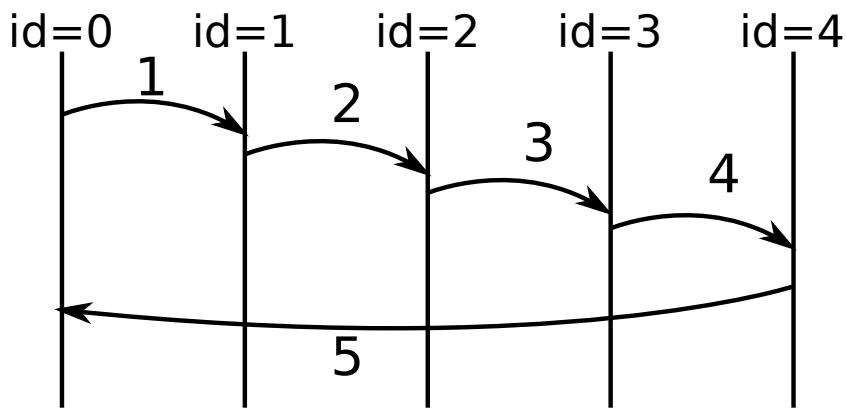


- Synchronous Message Passing: MPI

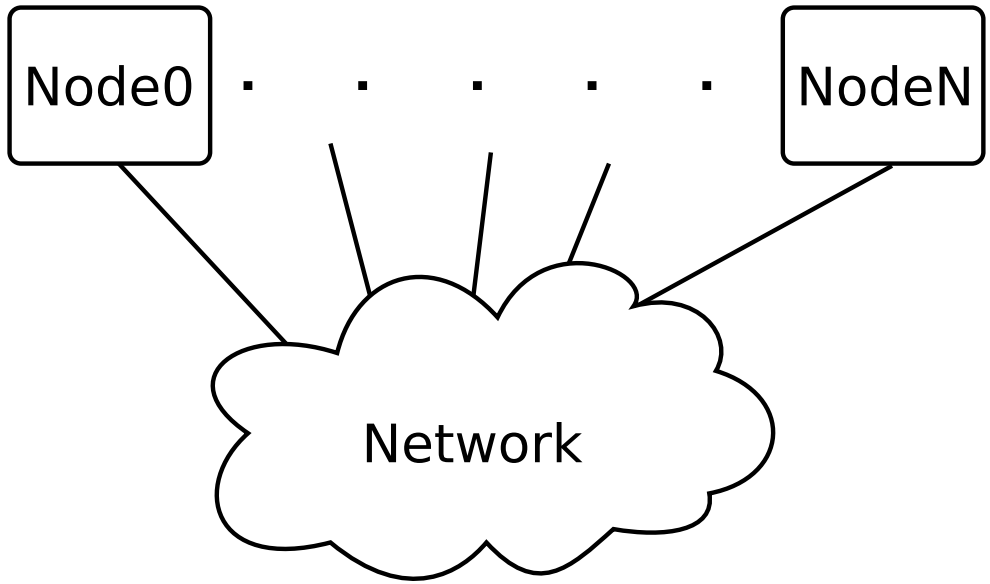
MPI

```c
int cnt = 1;

if (id == 0) {
 MPI_Send(&cnt, 1, MPI_INT, id+1, 0, MPI_COMM_WORLD);
 MPI_Recv(&cnt, 1, MPI_INT, N-1, 0, MPI_COMM_WORLD, NULL);
 printf("Counter: %d\n", cnt);
}
else {
 MPI_Recv(&cnt, 1, MPI_INT, id-1, 0, MPI_COMM_WORLD, NULL);
 cnt += 1;
 MPI_Send(&cnt, 1, MPI_INT, (id+1) % N, 0, MPI_COMM_WORLD);
}
```

```
% mpicc test.c
% mpirun -n 5 --hostfile hosts ./a.out
Counter: 5
```

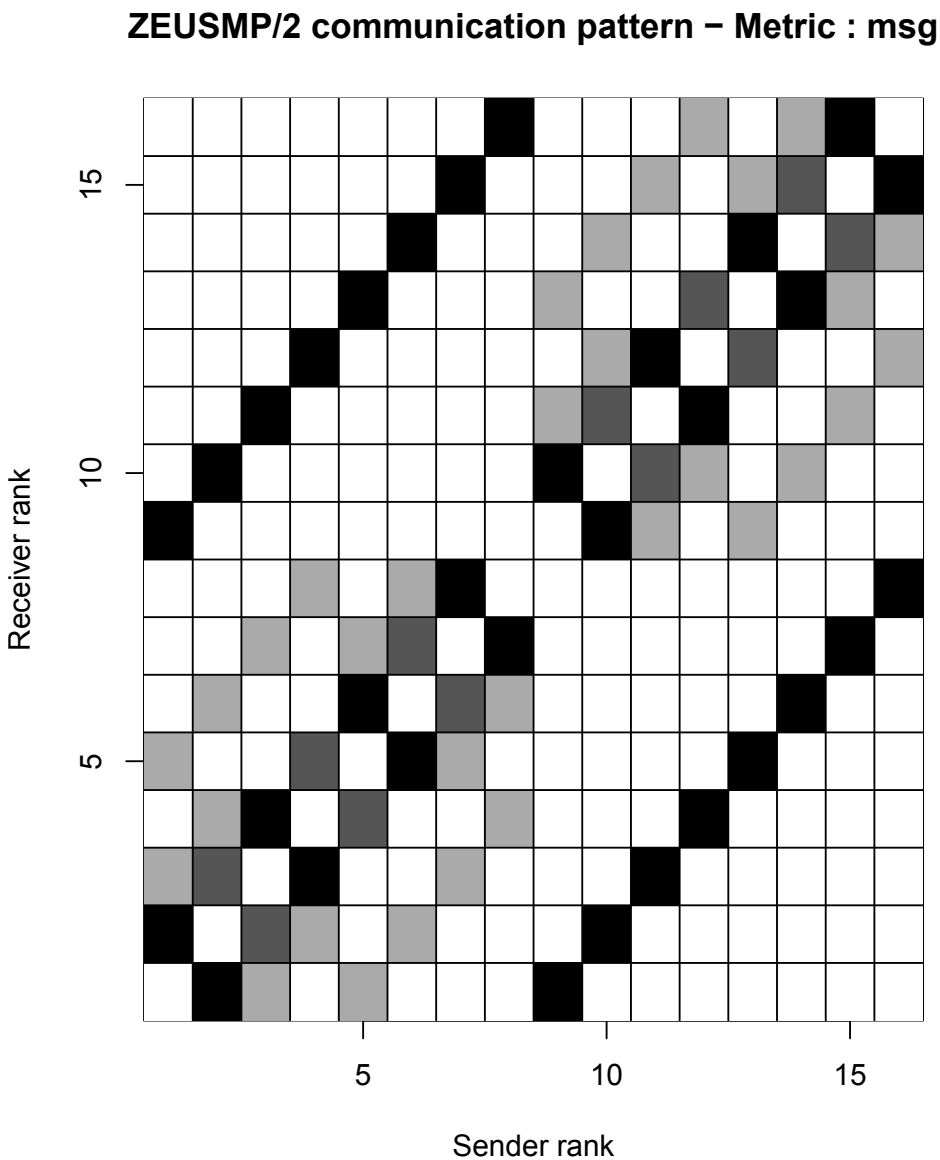


Clusters



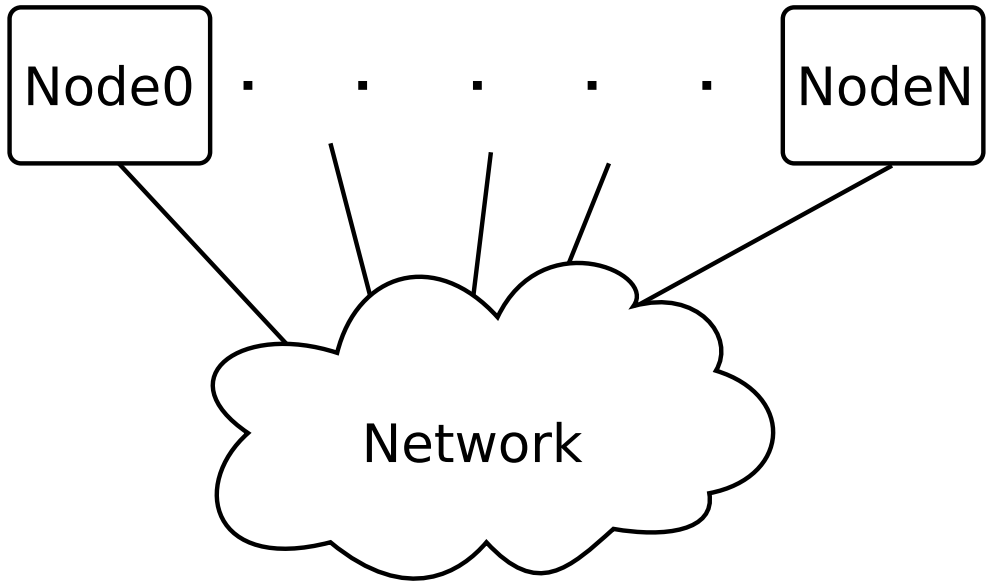
- Synchronous Message Passing: MPI
  - Process placement issue
    - Depends on network topology
    - e.g. TreeMatch

TreeMatch



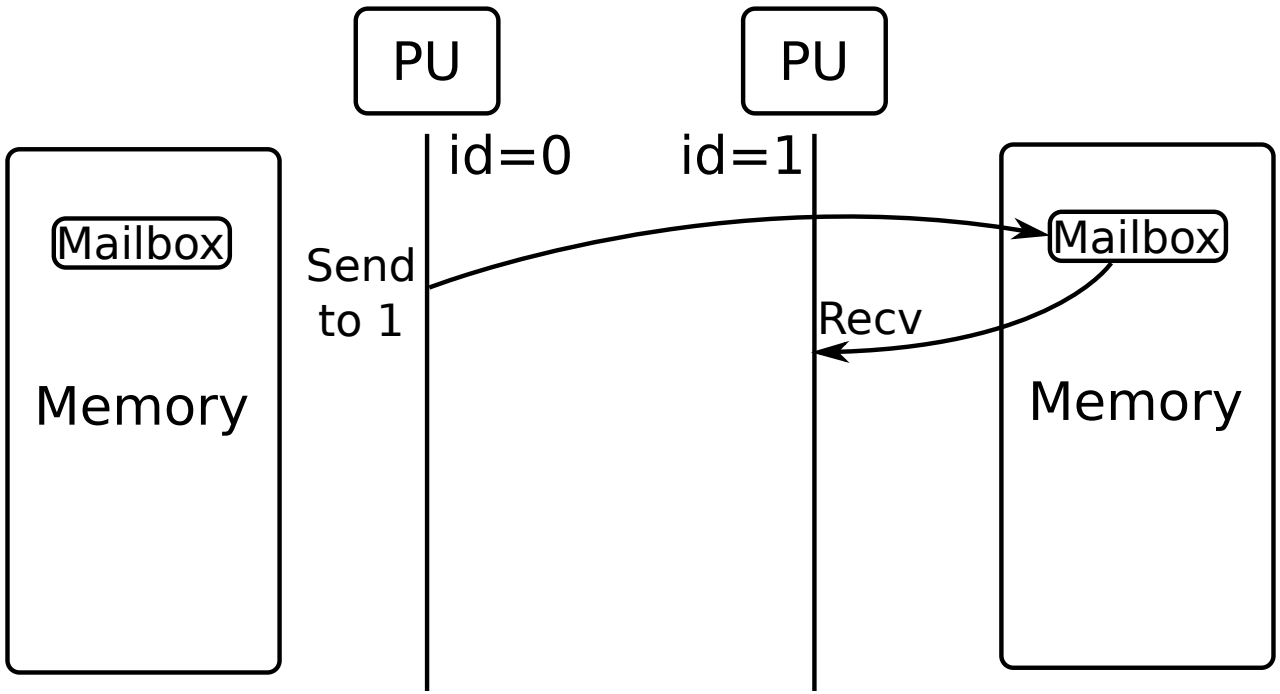
- Trace communications between processes
- Algorithm to map processes to network topology

Clusters



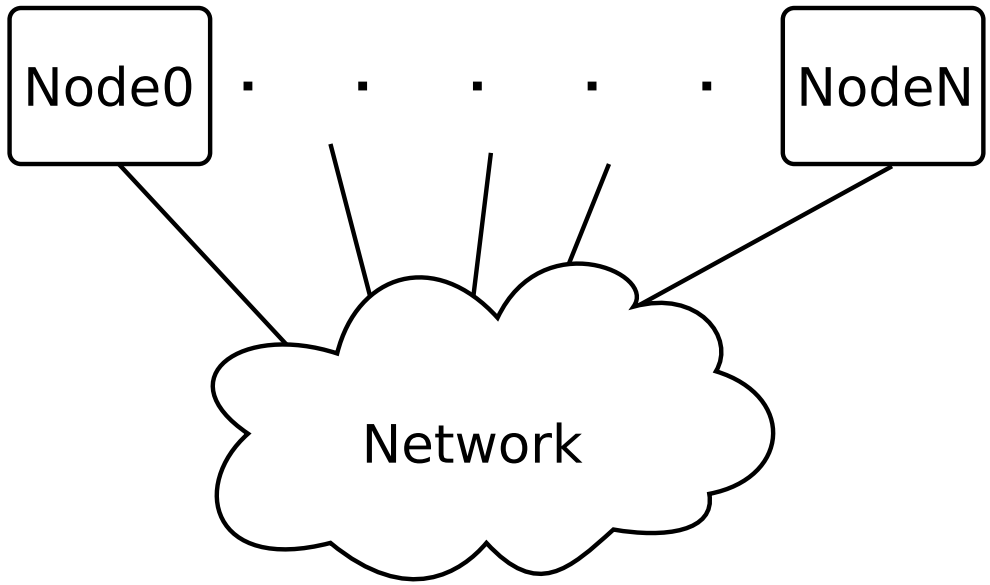
- Synchronous Message Passing: MPI
- Actor model / Active Objects: Charm++, Erlang...

Actor model



- Asynchronous message passing
- Dynamic actor creation

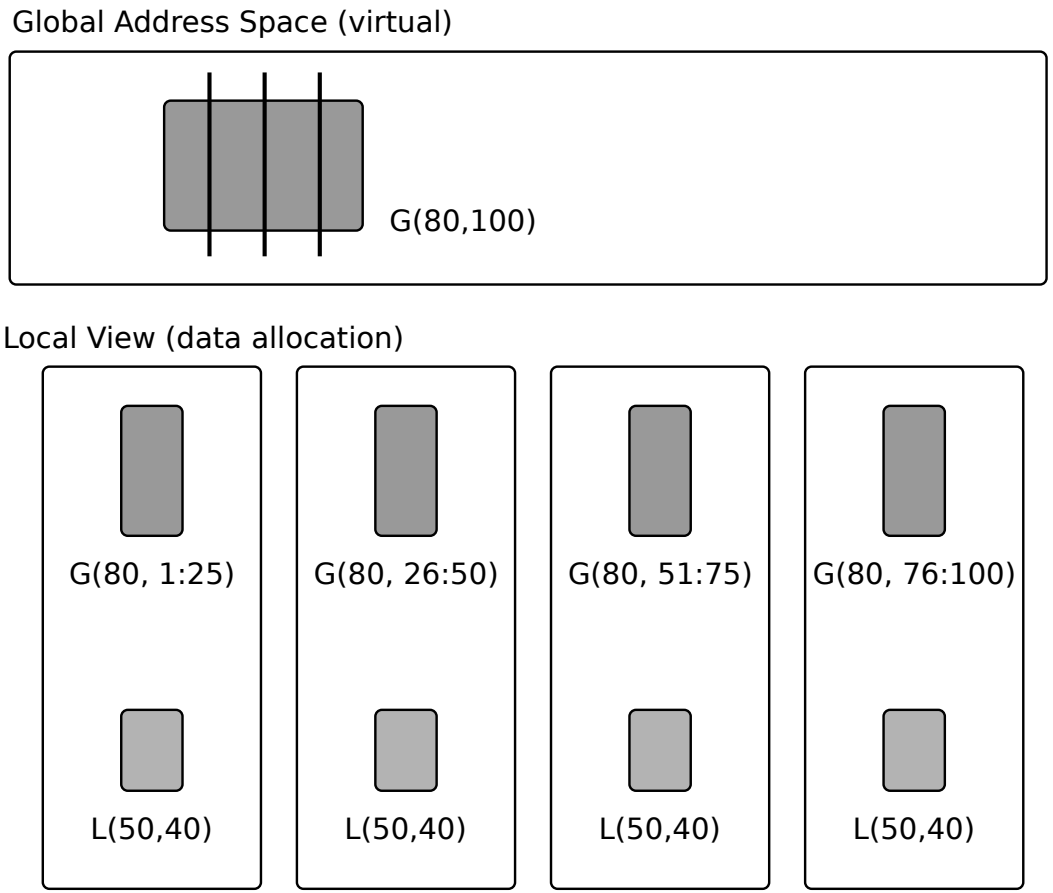
Clusters



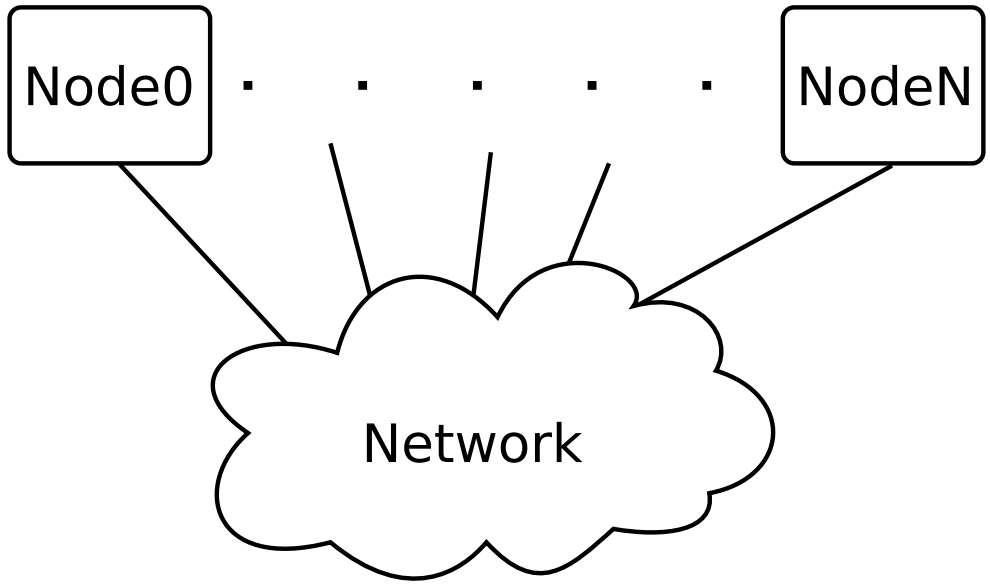
- Synchronous Message Passing: MPI
- Actor model / Active Objects: Charm++, Erlang...
- PGAS: HPF, XcalableMP, x10, ZPL/Chapel...
  - PGAS = Partitioned Global Address Space

PGAS: XcalableMP

```
!$xmp nodes P(4)
!$xmp template T(100)
!$xmp distribute T(block) onto P
      real G(80, 100)              ! global variable
!$xmp align G(*, i) with T(i)
      real L(50, 40)               ! local variable (default)
```

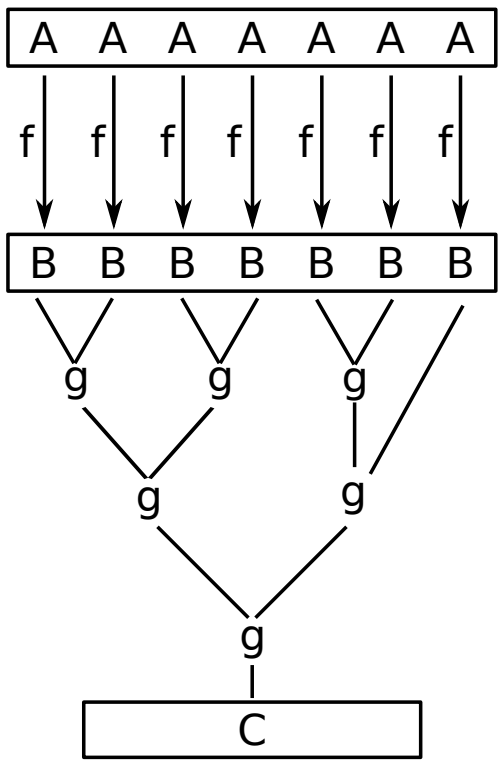


Clusters



- Synchronous Message Passing: MPI
- Actor model / Active Objects: Charm++, Erlang...
- PGAS: HPF, XcalableMP, x10, ZPL/Chapel...
- MapReduce: Google MapReduce, Hadoop...

MapReduce



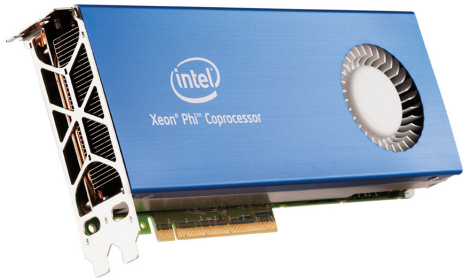
- reduce g . map f
  - Most operations on collections can be written this way
  - Easily parallelizable
- Users only have to provide f and g
- The runtime system provides
  - Task distribution
  - Communications
  - Fault tolerance
  - ...

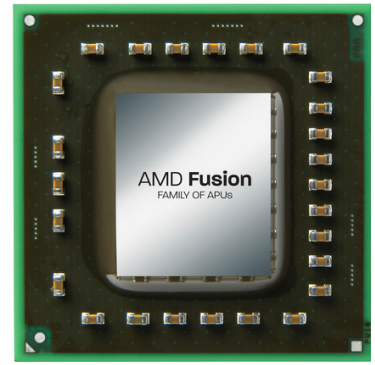
Keywords: Bird-Meertens Formalism, Catamorphism

Summary

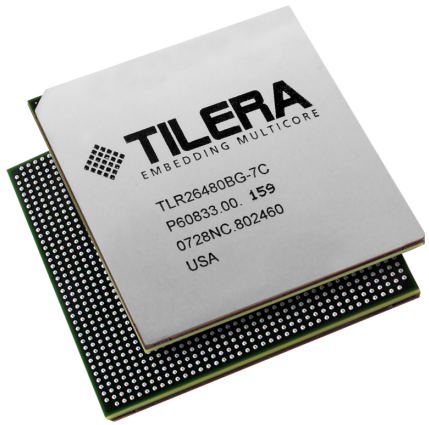
Architecture	Programming models
Multicore (NUMA)	OpenMP, PThread...
Cluster	MPI, PGAS, Actors, MapReduce...

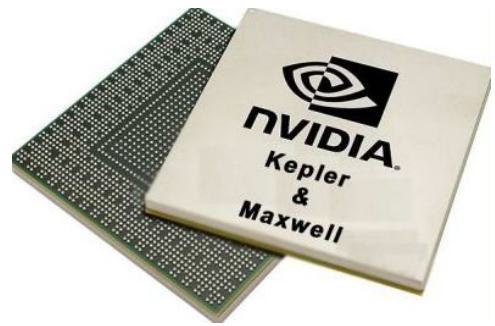
- They all assume homogeneous architectures
  - Same (or similar) architecture duplicated





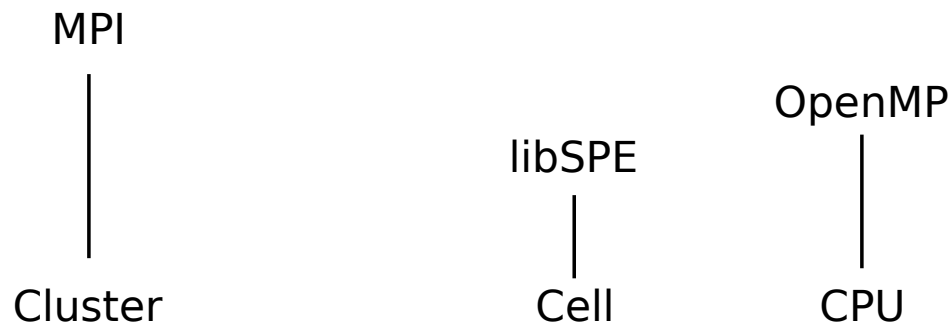
Accelerators & Many-Core



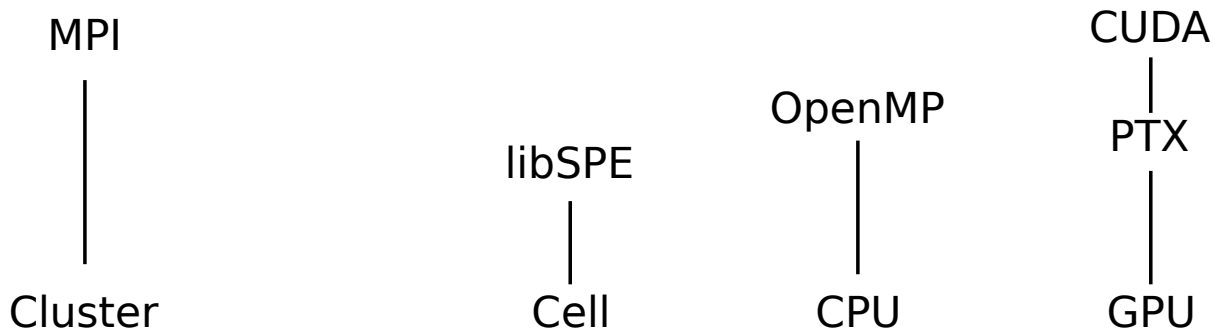


Frameworks...

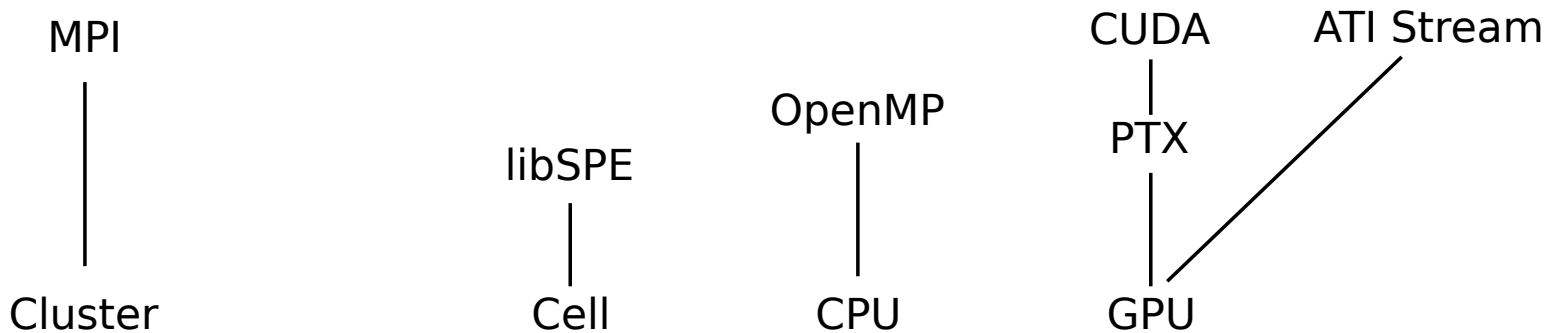
Frameworks...



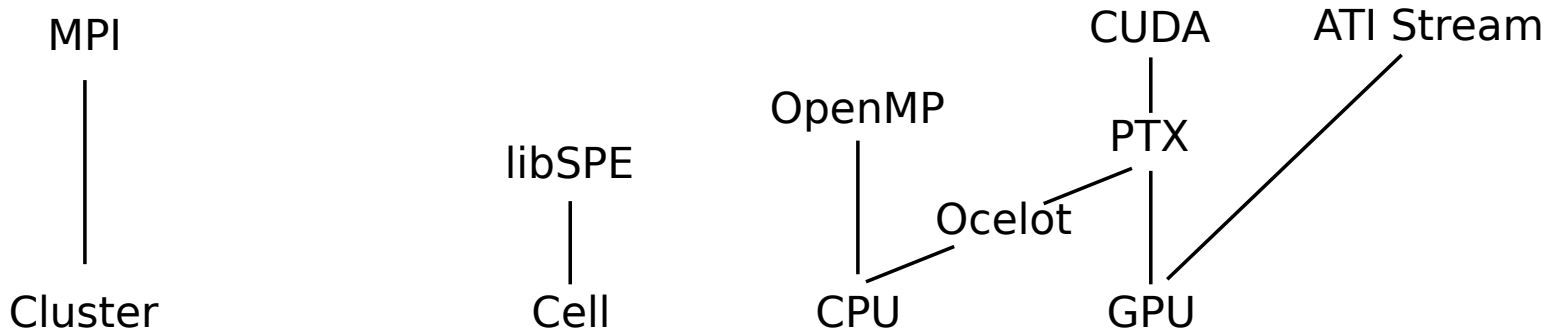
Frameworks...



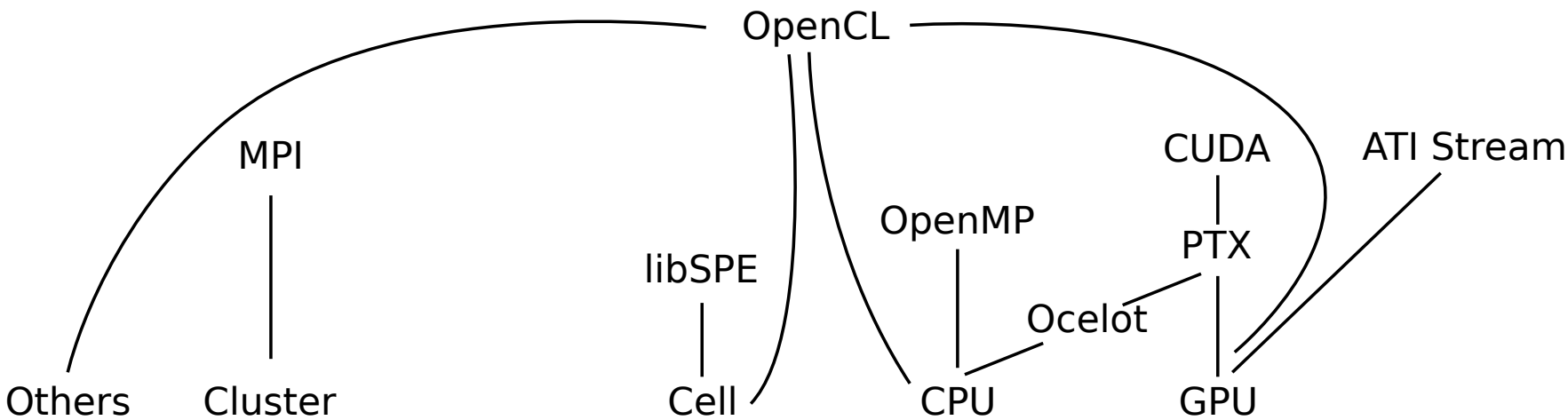
Frameworks...



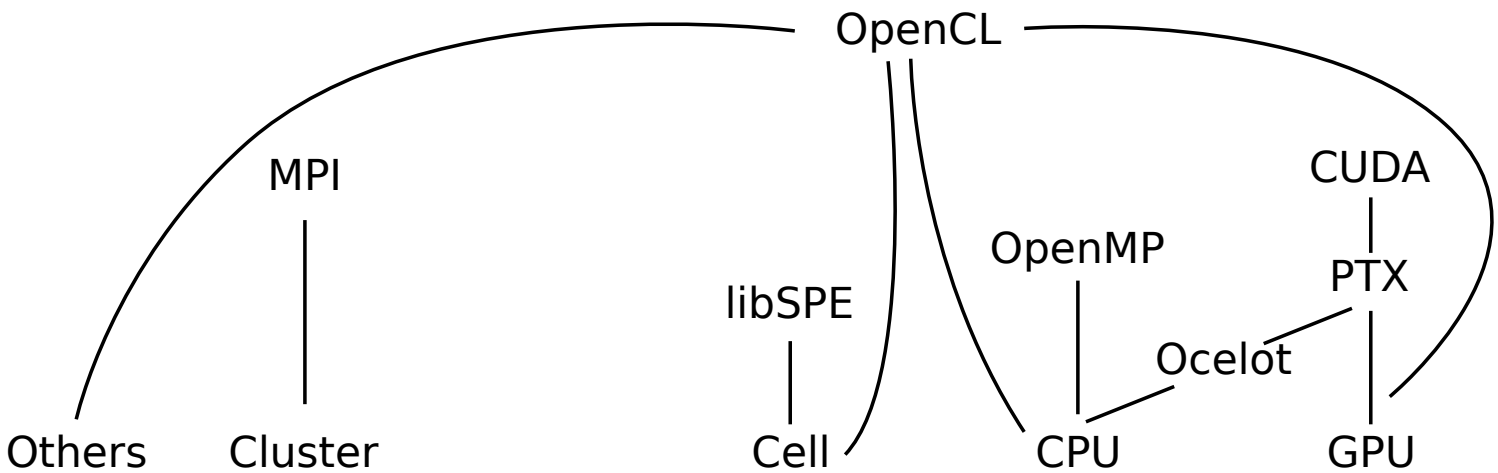
Frameworks...


MPI
Cluster
libSPE
Cell
OpenMP
CPU
CUDA
PTX
Ocelot
ATI Stream
GPU

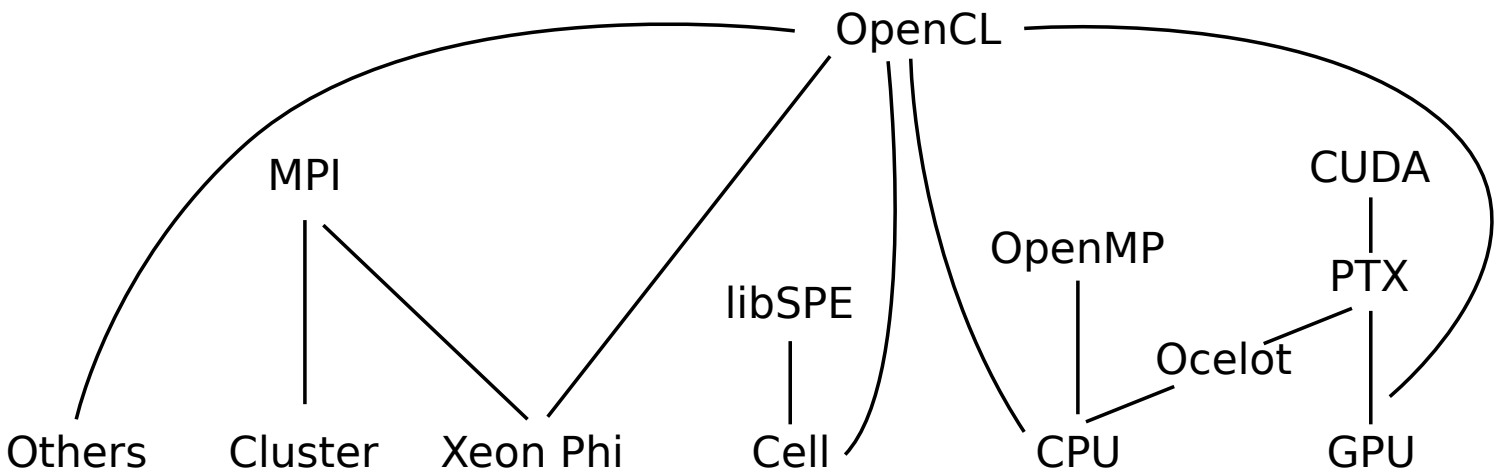
Frameworks...


OpenCL
MPI
CUDA
ATI Stream
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Cell
CPU
GPU

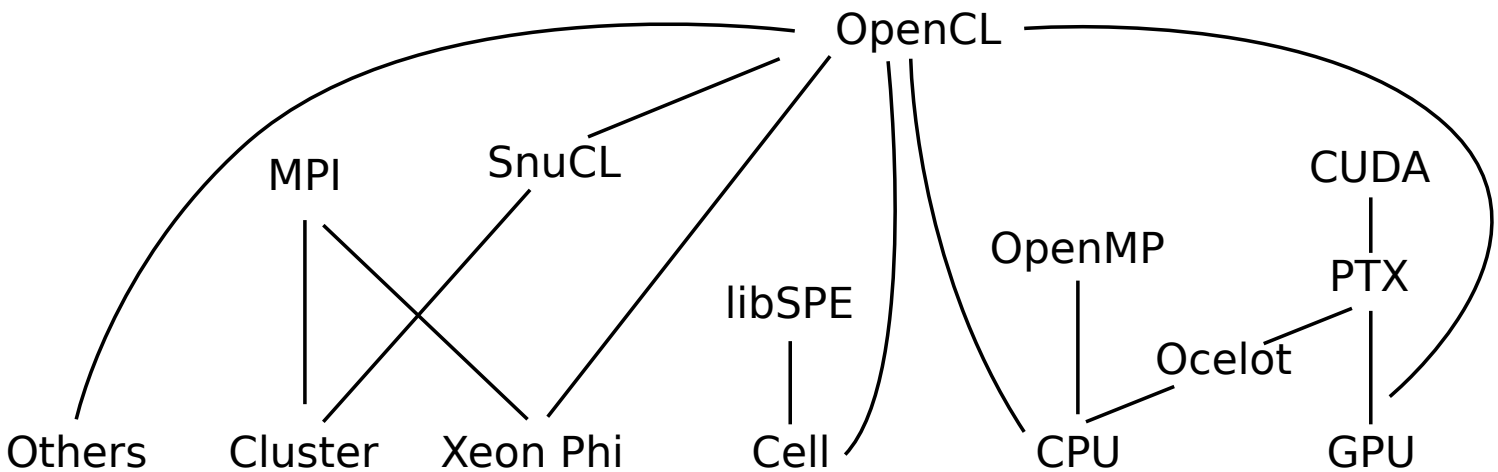
Frameworks...


OpenCL
MPI
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Cell
CPU
GPU

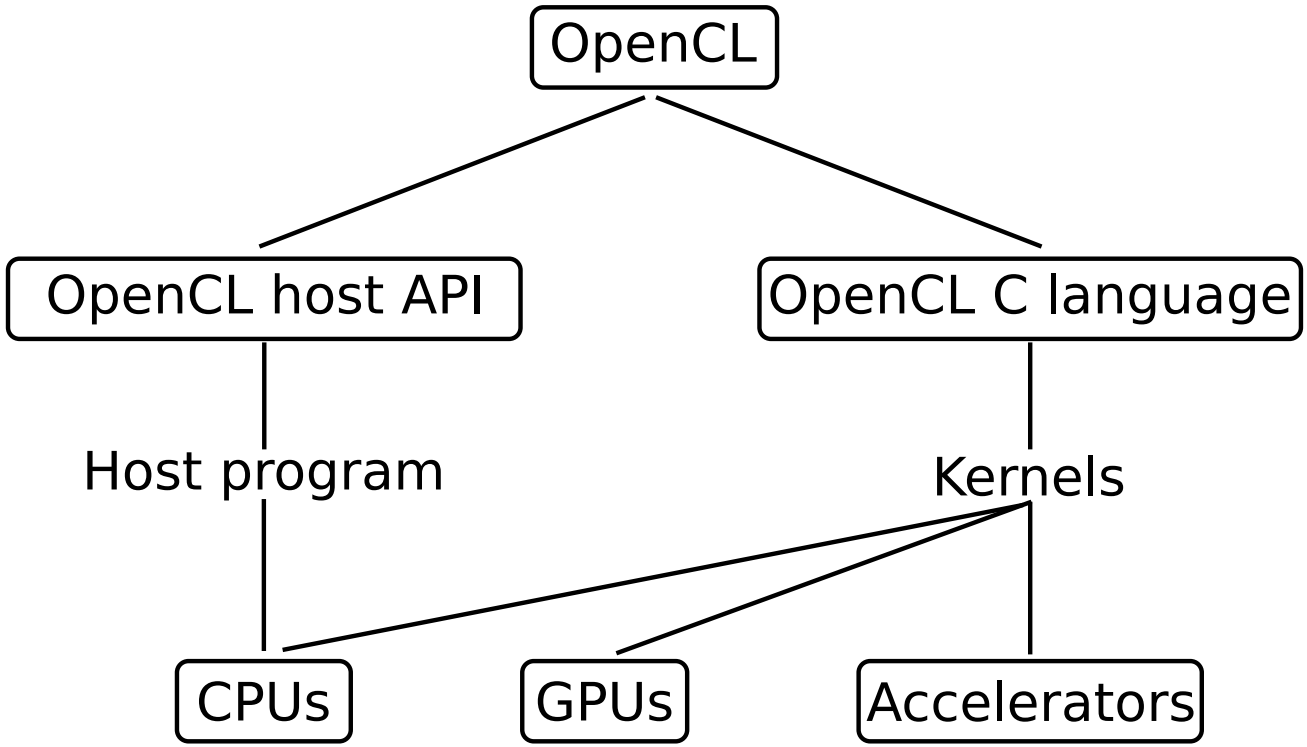
Frameworks...


OpenCL
MPI
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

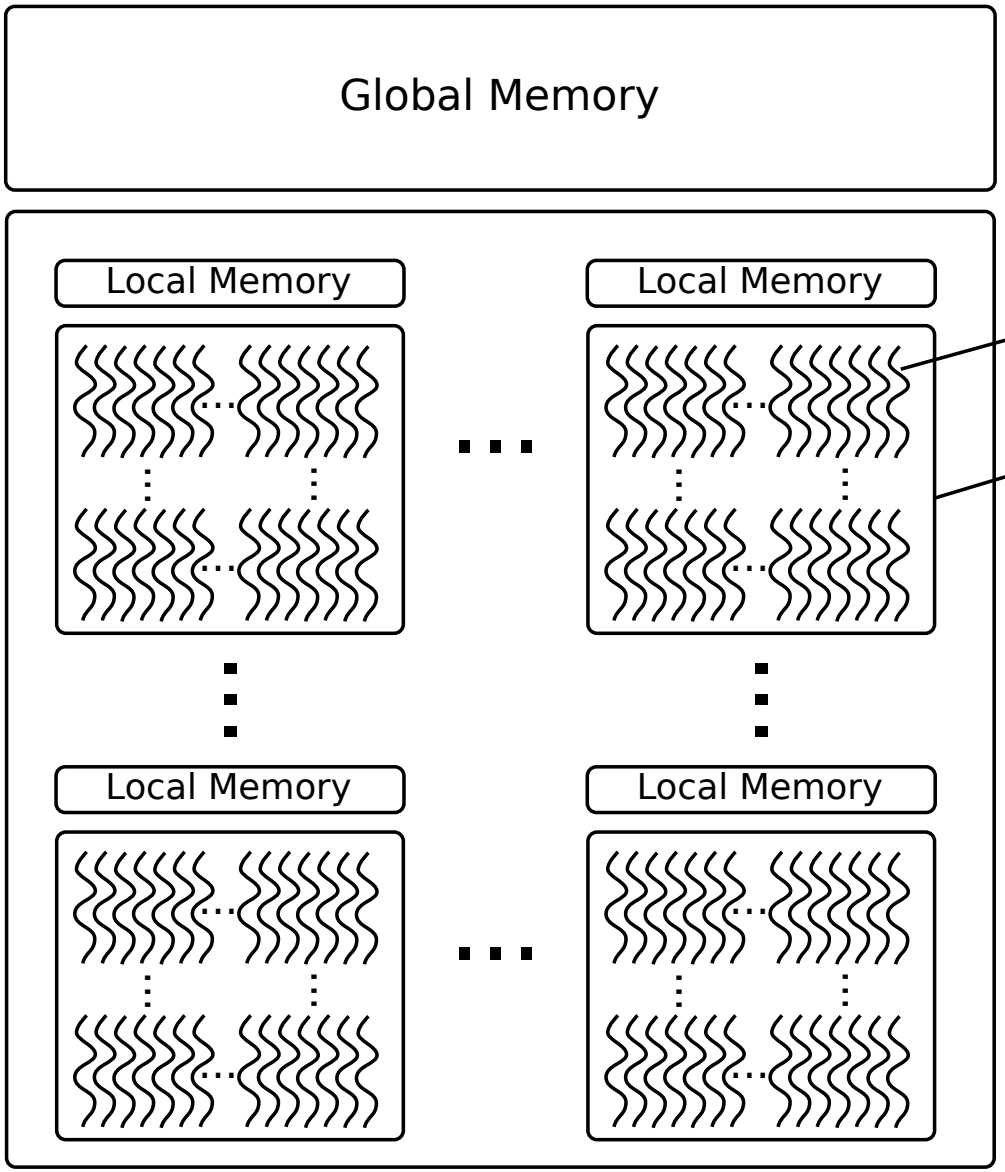
Frameworks...


OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

OpenCL


OpenCL
OpenCL host API
OpenCL C language
Host program
Kernels
CPUs
GPUs
Accelerators

OpenCL C Language



- SPMD
- get_local_id(int dim)
- get_global_id(int dim)

- __local, __global
- work-group barrier

OpenCL Host API

Device management

- Query available devices and their capabilities

Memory management

- Allocate/release buffers in device memory
- Transfer data between host memory and devices

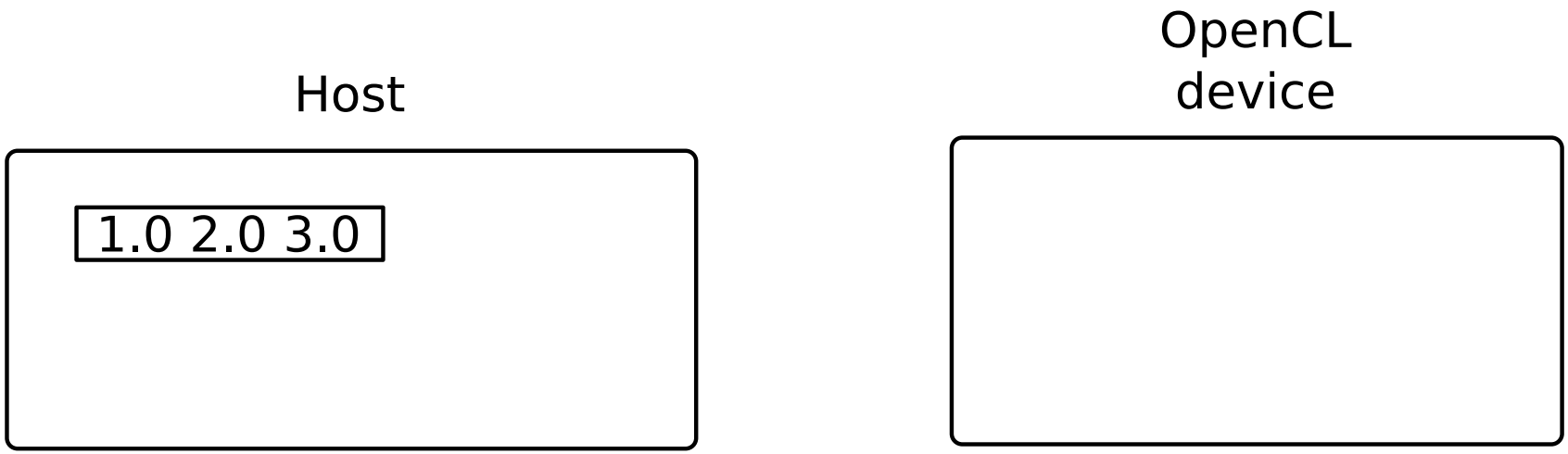
Kernel management

- Compilation
- Scheduling on devices

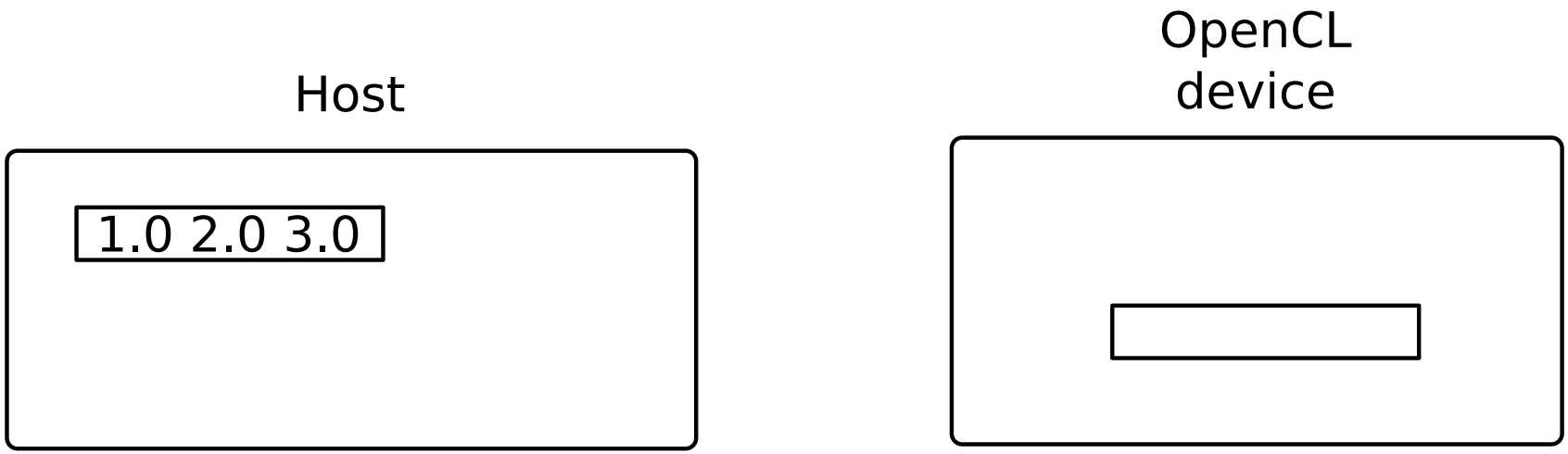
Synchronizations

- Command graph: events...
- With the host program: barriers...

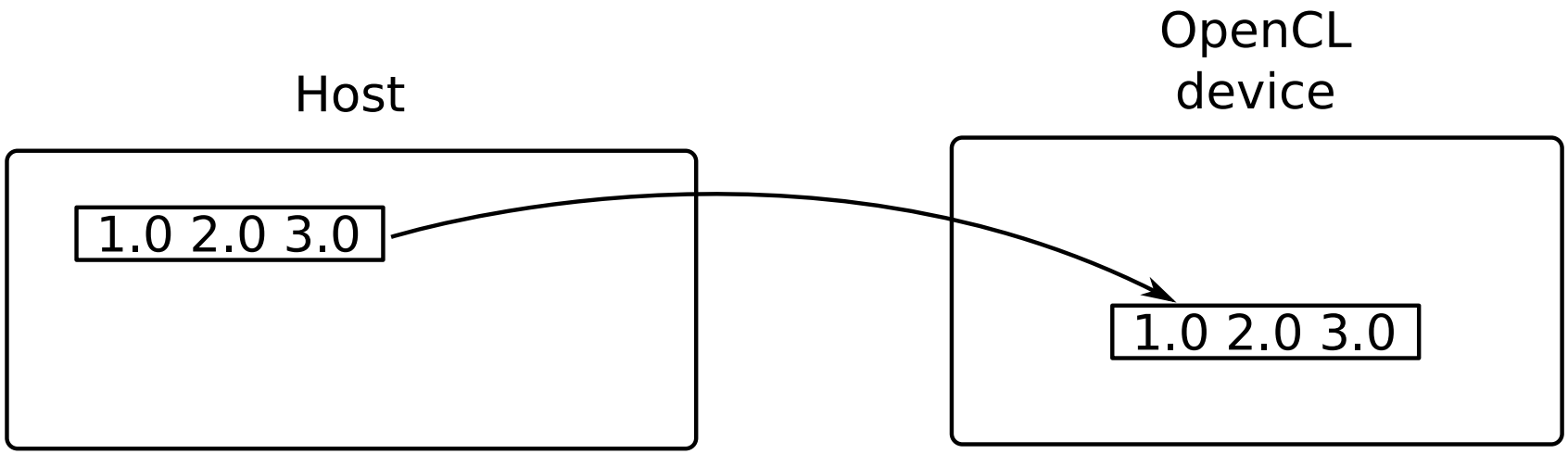
OpenCL example



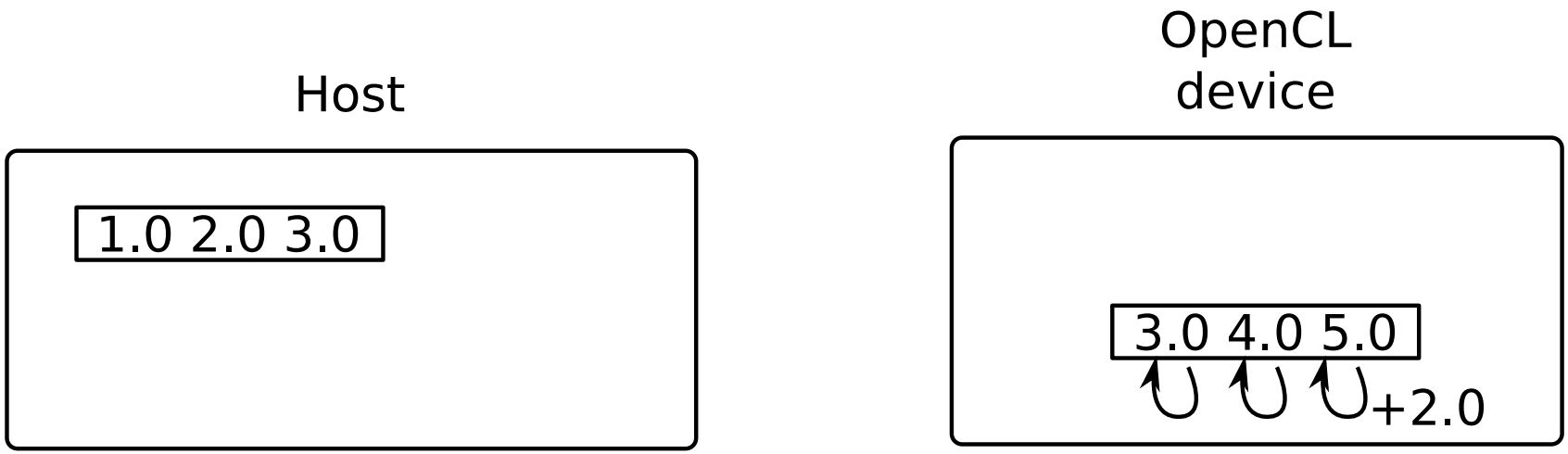
OpenCL example



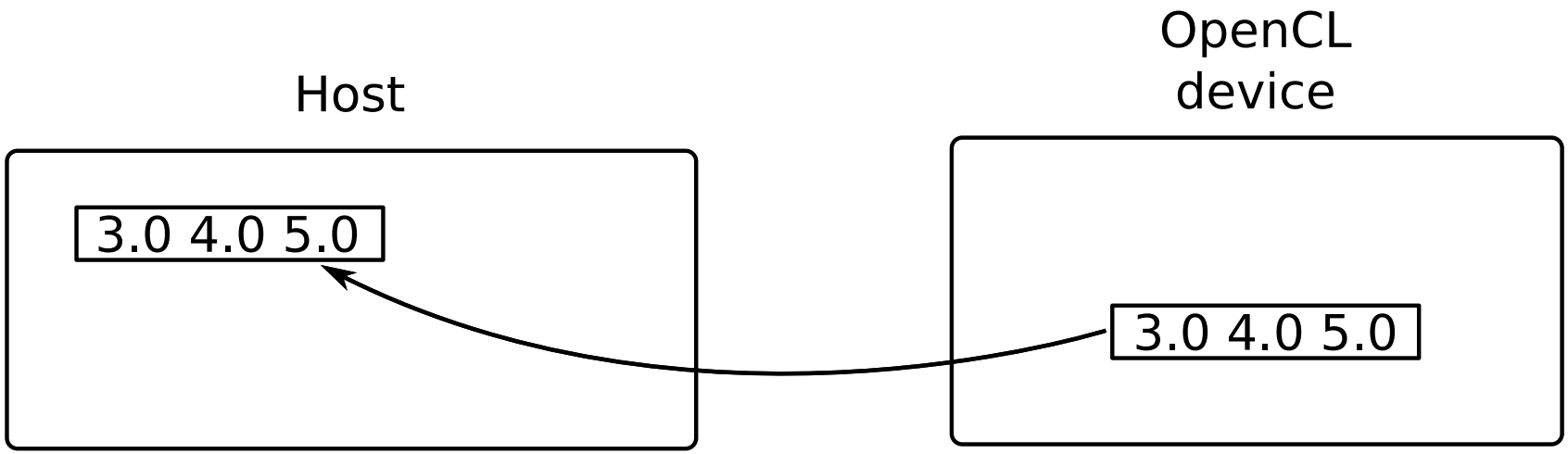
OpenCL example



OpenCL example



OpenCL example



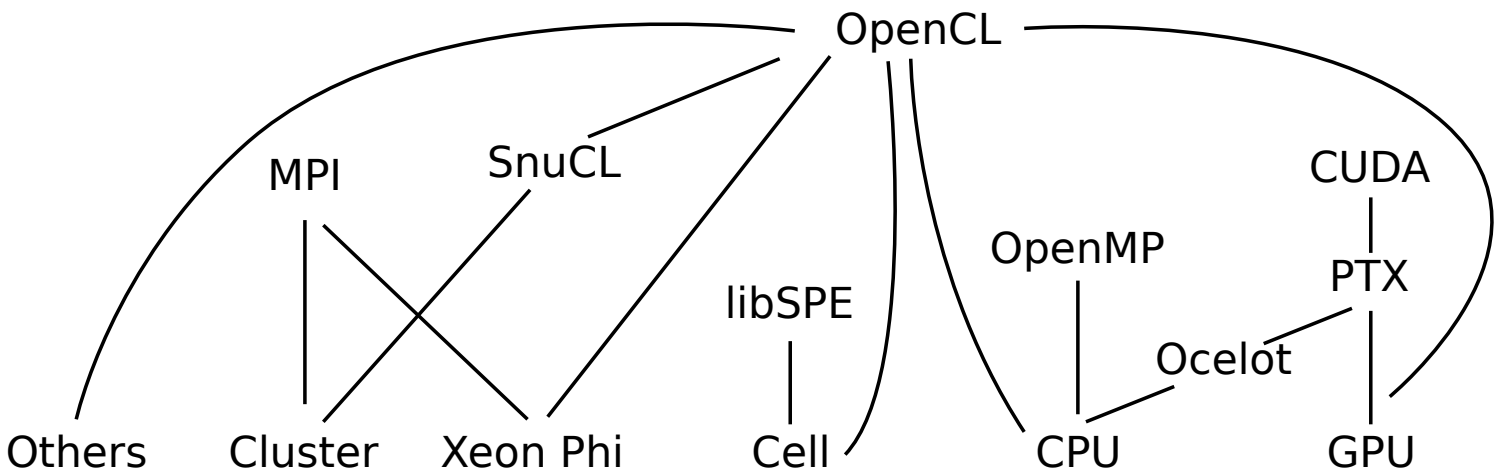
OpenCL example: kernel

```
__kernel void add2(__global float *out)
{
    size_t id = get_global_id(0);

    out[id] += 2.0;
}
```

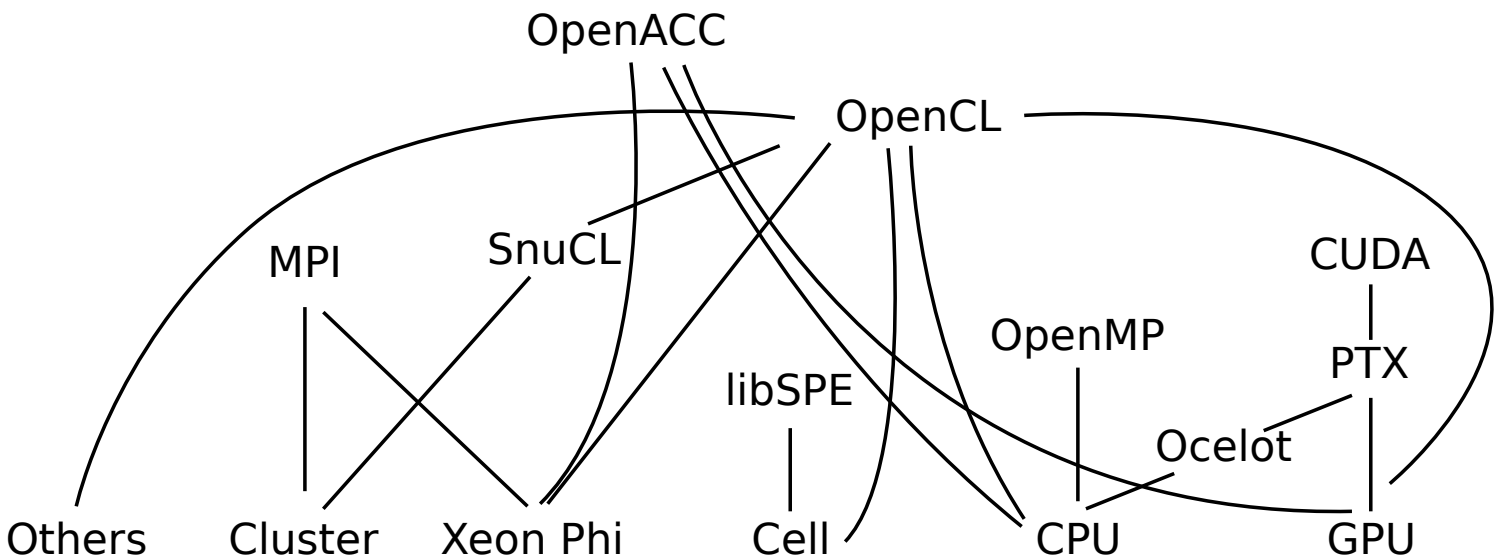
OpenCL example: host API

```
float data[256] = {...};

clGetPlatformIDs(max_pfs, pfs, NULL);
clGetDeviceIDs(pfs[0],CL_DEVICE_TYPE_ALL,1,devs,NULL);
ctx = clCreateContext(NULL, 1, devs[0], NULL,NULL,NULL);
cq = clCreateCommandQueue(ctx, devs[0], 0, NULL);

buf = clCreateBuffer(ctx,CL_MEM_READ_WRITE,1024,NULL,NULL);

clEnqueueWriteBuffer(cq,buf,CL_FALSE,0,1024,data,0,NULL,NULL);

program = clCreateProgramWithSource(ctx,1,&src,NULL,NULL);
clBuildProgram(program, 0, NULL, NULL, NULL, NULL);
ker = clCreateKernel(program, "add2", NULL);

clSetKernelArg(ker, 0, sizeof(cl_mem), &buf);
size_t group[3] = {32, 1, 1};
size_t grid[3] = {256, 1, 1};
clEnqueueNDRangeKernel(cq,ker,3,NULL,grid,group,0,NULL,NULL);

clEnqueueReadBuffer(cq,buf,CL_FALSE,0,1024,data,0,NULL,NULL);

clFinish(cq);
```

Frameworks...


OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

Frameworks...


OpenACC
OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

OpenACC

Concepts

- Generate kernels from annotated C/Fortran code
- Simpler memory model (one accelerator)
- Ensure memory coherence with annotations

OpenACC example

```
float data[256] = {...};

#pragma acc kernels copy(data)
for (int i=0; i<256; i++)
    data[i] += 2.0;
```

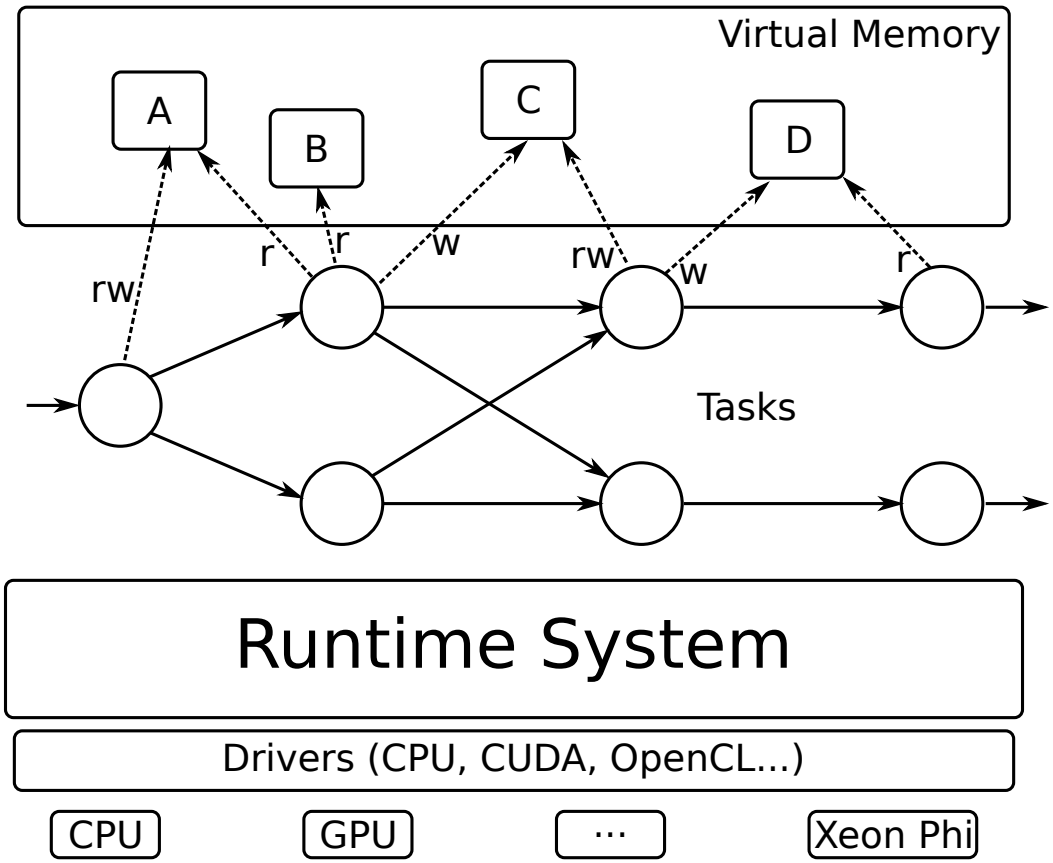
Limitations

Low-level approaches

- No support for multi-accelerator (OpenACC)
- Hard to manage several accelerators (OpenCL)

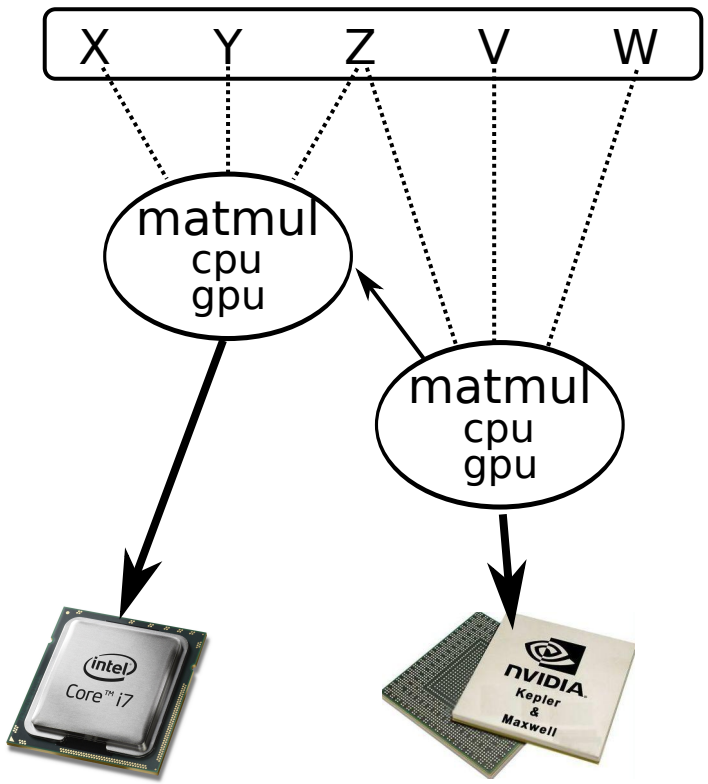
Runtime Systems

Runtime Systems



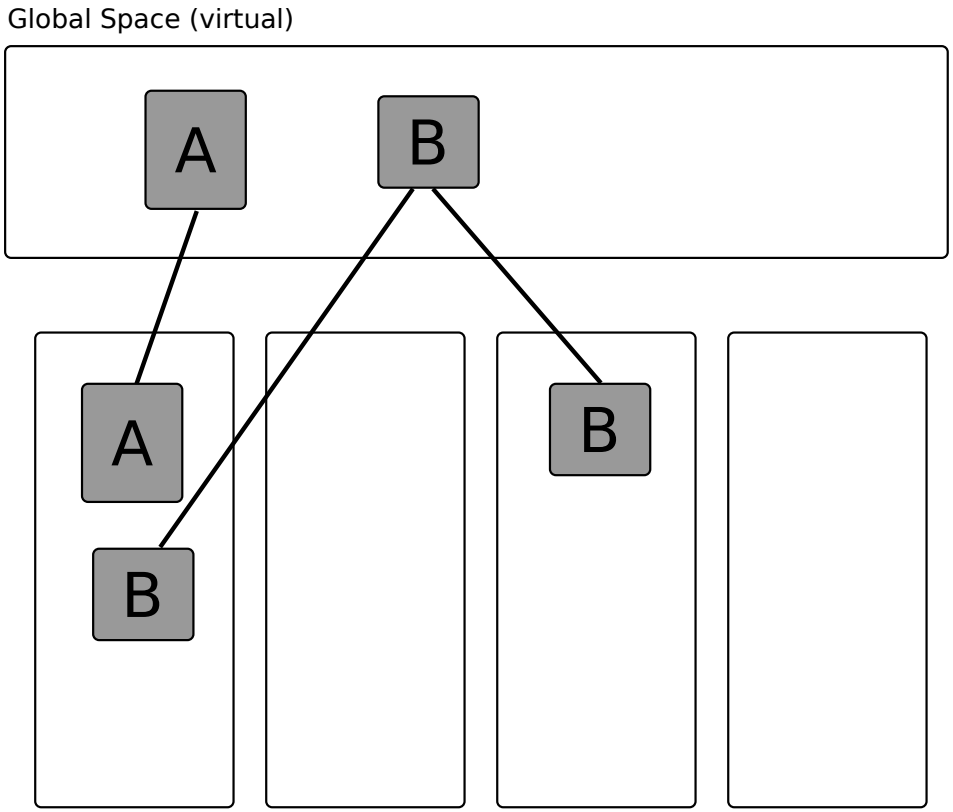
- Automatically schedule kernels on available accelerators
- Automatically manage distributed memory
- e.g. StarSS, StarPU...

Runtime Systems



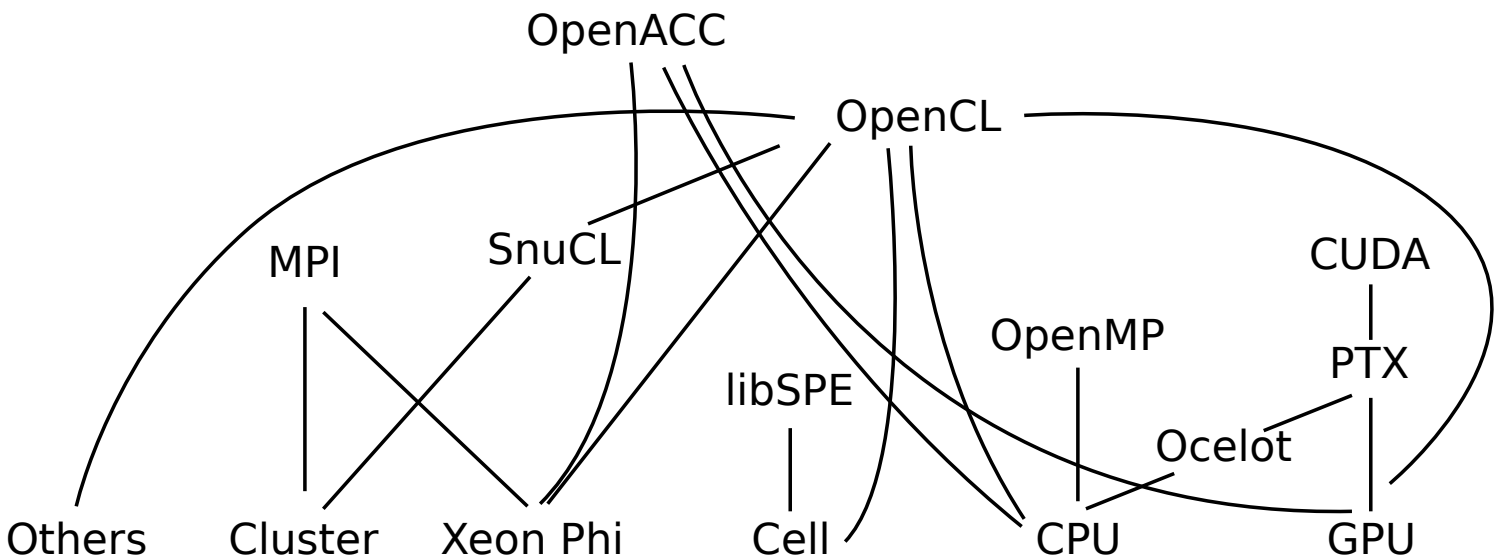
- Automatically schedule kernels on available accelerators
- Automatically manage distributed memory
- e.g. StarSS, StarPU...

Runtime Systems

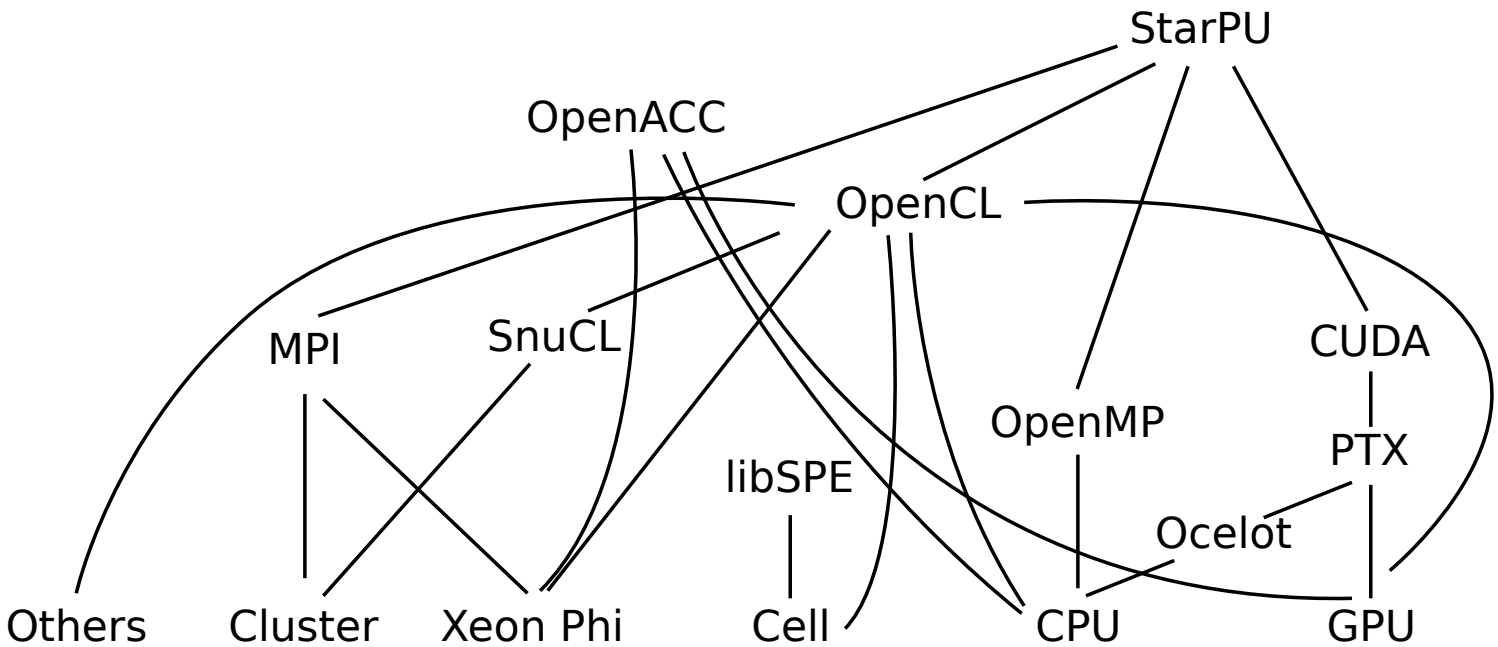


- Automatically schedule kernels on available accelerators
- Automatically manage distributed memory
- e.g. StarSS, StarPU...

Frameworks...


OpenACC
OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

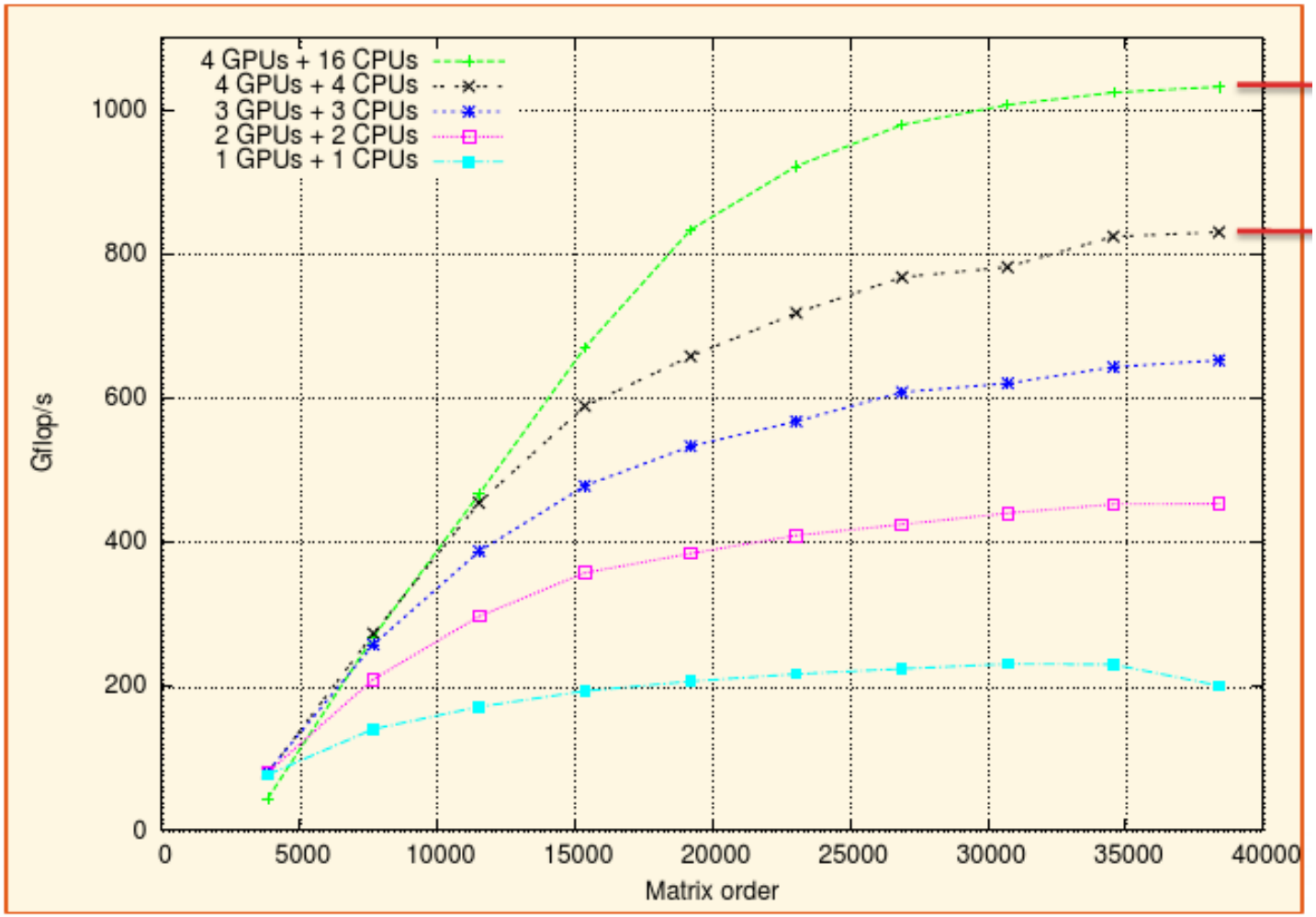
Frameworks...


StarPU
OpenACC
OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

StarPU example

```
void scal_cpu_func(void * buffers[], vois *args);
void scal_cuda_func(void * buffers[], vois *args);

starpu_codelet scal_cl = {
    .where = STARPU_CPU | STARPU_CUDA,
    .cpu_func = {scal_cpu_func, NULL},
    .cuda_func = {scal_cuda_func, NULL},
    .nbuffers = 1,
    .modes = {STARPU_RW}
}

float vector[N];

starpu_data_handle vector_handle;

starpu_vector_data_register(&vector_handle, 0,
          vector, N, sizeof(float));

starpu_insert_task(&scal_cl, STARPU_RW, vector_handle, 0);

starpu_task_wait_for_all();
starpu_data_unregister(vector_handle);
```

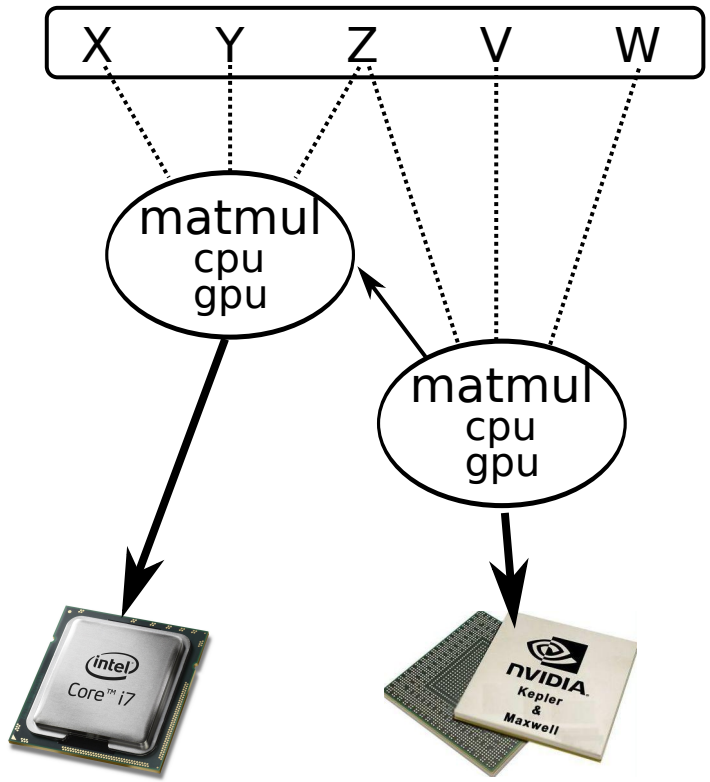
StarPU performance



UTK Magma (linear algebra) - QR factorization
16 CPUs (AMD) + 4 GPUs (C1060)

Limitations

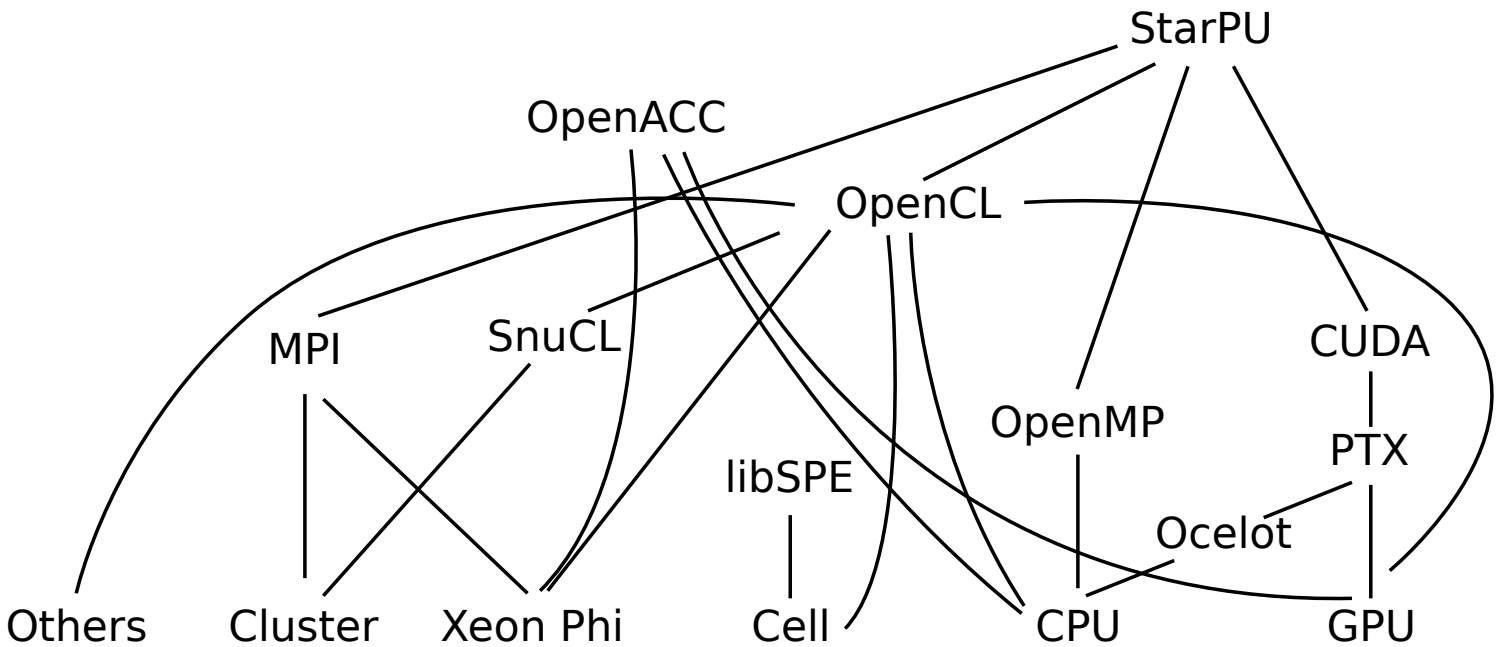
- Task/data granularity adaptation
- Task graph optimizations



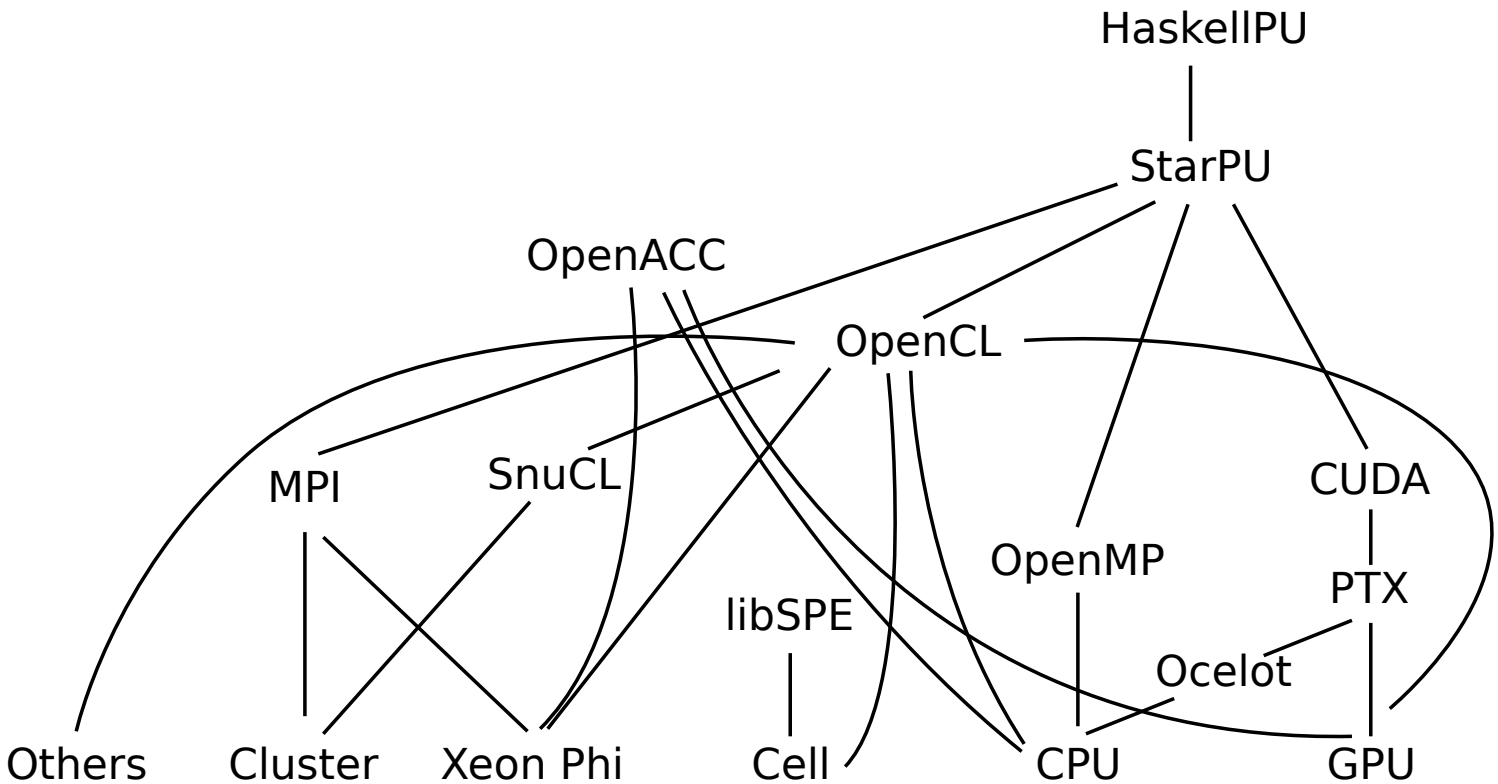
What I am working on

- Use a functional language to describe the task graph
- More amenable to optimizations
  - Granularity adaptation expected
- e.g. HaskellPU, ViperVM

Frameworks...


StarPU
OpenACC
OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

Frameworks...

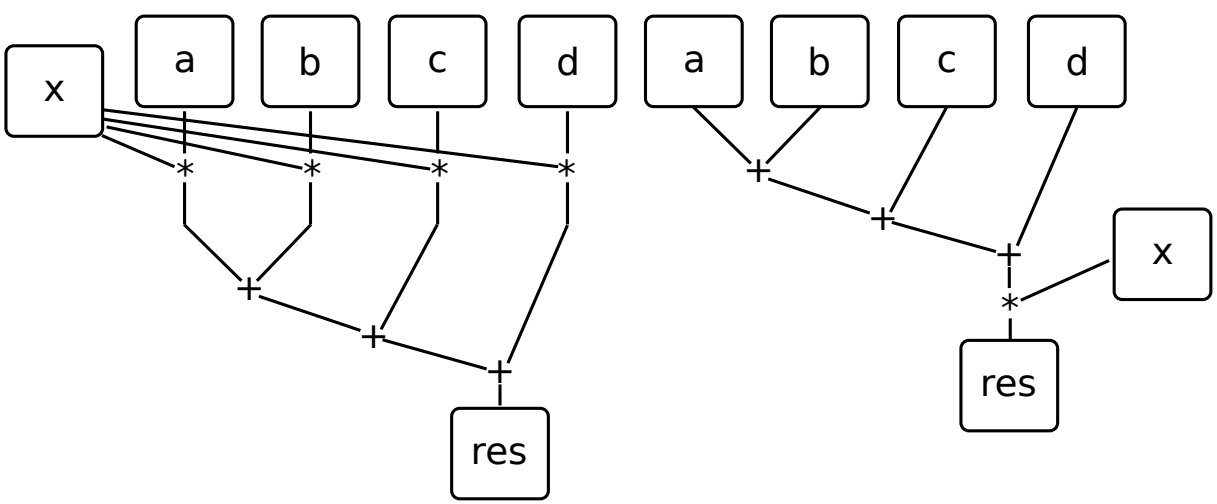

HaskellPU
StarPU
OpenACC
OpenCL
MPI
SnuCL
CUDA
OpenMP
PTX
libSPE
Ocelot
Others
Cluster
Xeon Phi
Cell
CPU
GPU

HaskellPU

```
let res = a*x + b*x + c*x + d*x
computeSync res
```

```
let res = (a + b + c + d) * x
computeSync res
```

```
let res = (reduce (+) [a,b,c,d]) * x
computeSync res
```



Conclusion

- Things are evolving quickly!
  - Architectures
  - Programming models
- Runtime systems are now accepted in the HPC community
  - Ensure portability and performance
- New challenges
  - Fault tolerance
  - Power management
  - Collaboration with the programming language community (FP...)

Thank you!