TECHNISCHE
UNIVERSITÄT
DARMSTADT

System and Parallel Programming

Prof. Dr. Felix Wolf

PARALLEL PROGRAMMING MODELS

Parallel programing model

TECHNISCHE UNIVERSITÄT DARMSTADT

Abstraction of the underlying computer system that allows for the expression of parallel algorithms and data structures

Adapted from McCormick et al.

↓

Implementation via

Languages or language extensions	APIs	Compiler directives

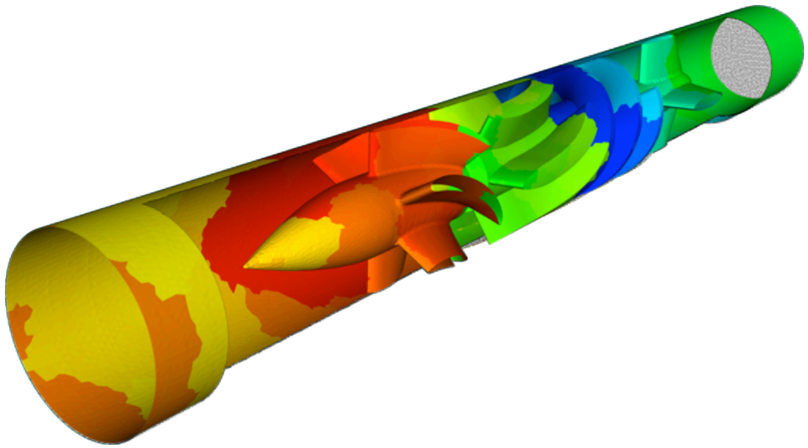
Objectives

Performance	Maximize parallel speedup
Productivity	Minimize time needed for • Writing code • Debugging • Performance optimization
Portability	Compiles & runs on another system Achieves comparable performance

Parallel programming model

Example

- Ventricular assist device
- FEM method
- Parallelization via geometric domain decomposition



Key abstractions

Concurrency

Memory

Communication

Synchronization

Single Program Multiple Data

- The same program is executed on multiple processors
- Underlying principle of most programming models
- Processes or threads are enumerated
- Each process or thread knows
- Its own number (ID)
- The total number

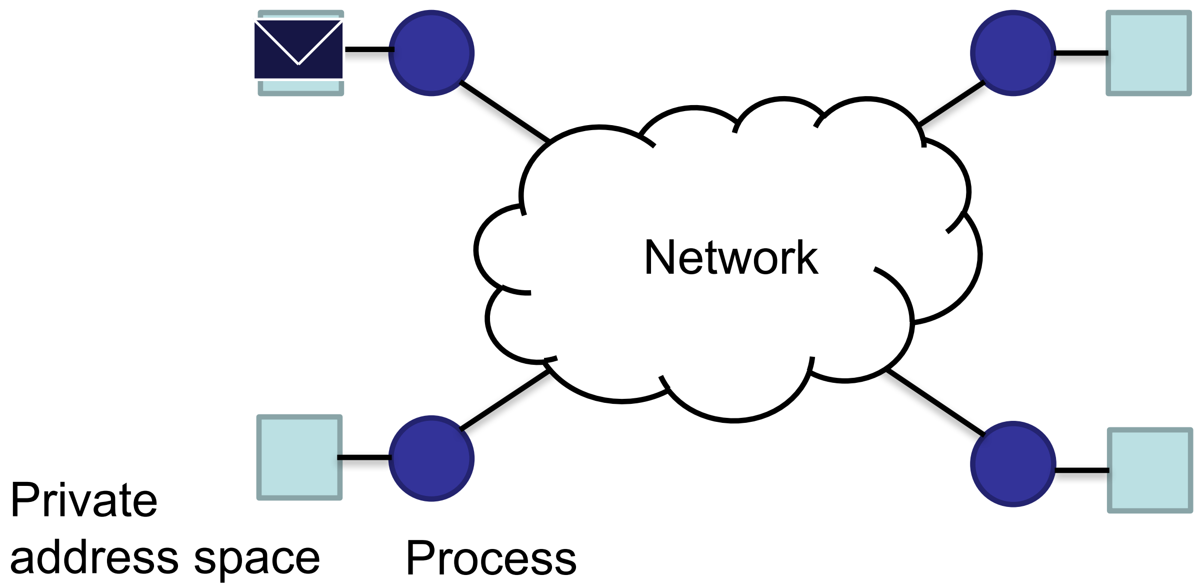
```
if (process_id == 42) then
  call do_something()
else
  call do_something_else()
endif
```

Same program but different control flows

Popular parallel programming models

Programming model	Primary target	Specific examples
Message passing	Compute cluster 	**MPI**
Multithreading	Multicore server 	**OpenMP**, C++11
GPGPU computing	GPU 	**CUDA**, OpenCL

Message Passing Interface



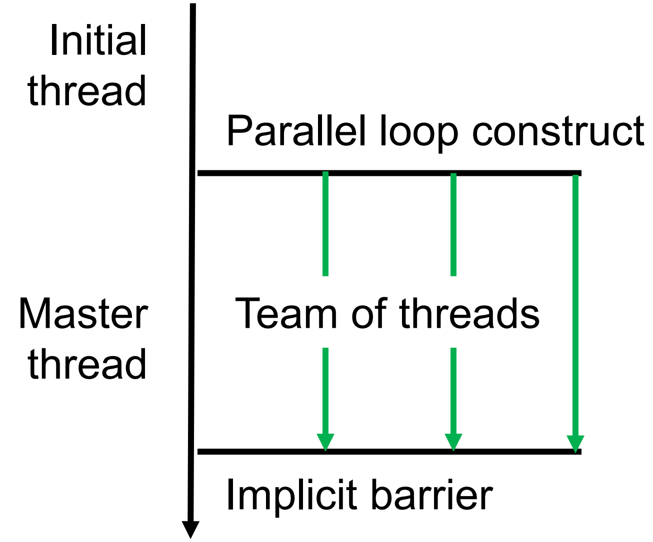

```
if (my_rank == SENDER)
   MPI_Send(buffer, count, datatype, RECEIVER, …);

if (my_rank == RECEIVER)
   MPI_Recv(buffer, count, datatype, SENDER, …);
```

OpenMP

```
void saxpy(…)
{
  int i;

#pragma omp parallel for
  for ( i = 0; i < n; i++ )
    z[i] = a * x[i] + y[i];
}
```

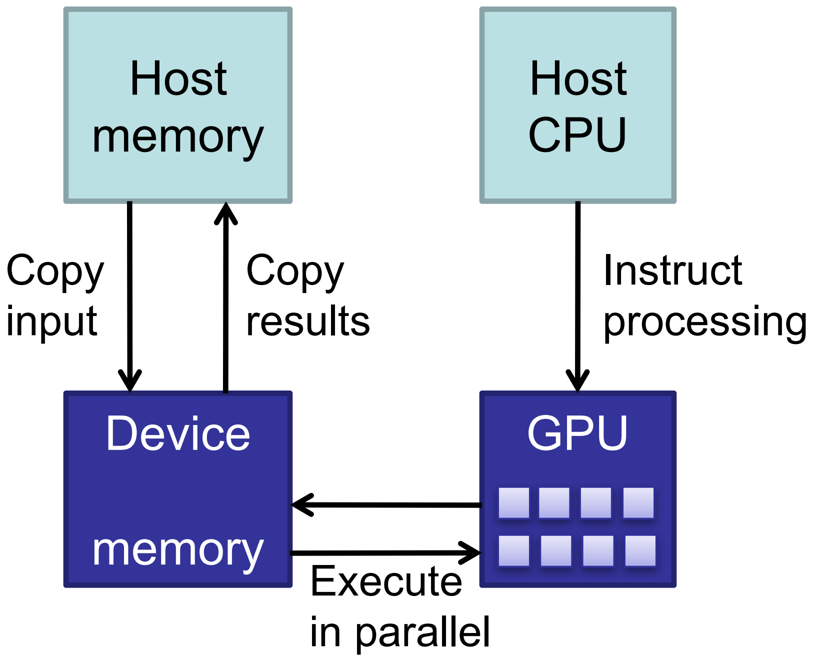


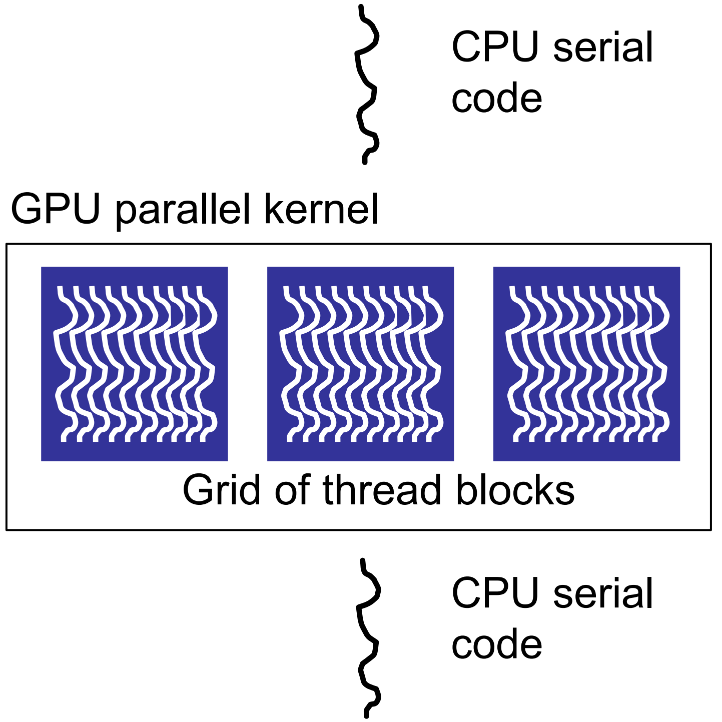
- Multiple threads communicate via shared variables
- Synchronization through barriers, lock-style methods, and atomic operations

CUDA

C with NVIDIA extensions

- Suitable for data-parallel workloads





Example: saxpy

Computing z = ax + y with serial loop

```
void saxpy_serial(…)
{
  int i;
  for (i=0; i<n; i++)
    z[i]= a * x[i] + y[i];
}
```

```
/* invoke serial saxpy kernel */
saxpy_serial(…);
```

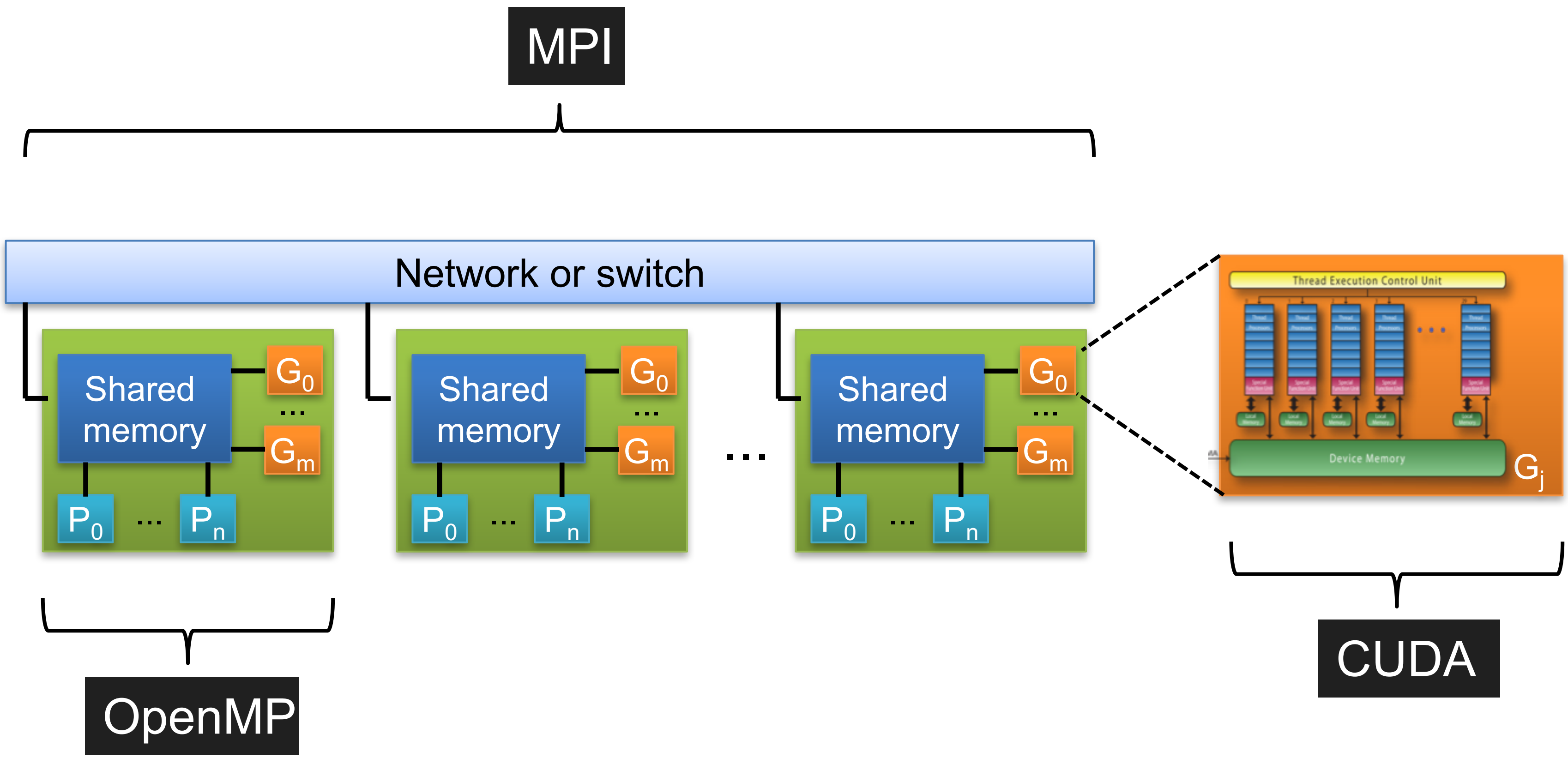
Example: saxpy (2)

Computing z = ax + y with parallel loop

```
__global__
void saxpy_parallel(…)
{
  int i = blockIdx.x * blockDim.x + threadIdx.x;
  if (i<n) z[i]= a * x[i] + y[i];
}
```

```
/* invoke parallel saxpy kernel with n threads */
/* organized in 256 threads per block */
int nblocks = (n + 255) / 256;
saxpy_parallel<<<nblocks, 256>>>(…);
```

Hybrid programming: MPI + X



Comparison

Programming model	Advantage	Disadvantage
MPI	Scalable	Parallelization requires major re-design
OpenMP	Incremental parallelization	Limited scalability Hard to debug
CUDA	Efficient & scalable for data-parallel workloads	High code complexity, laborious optimization