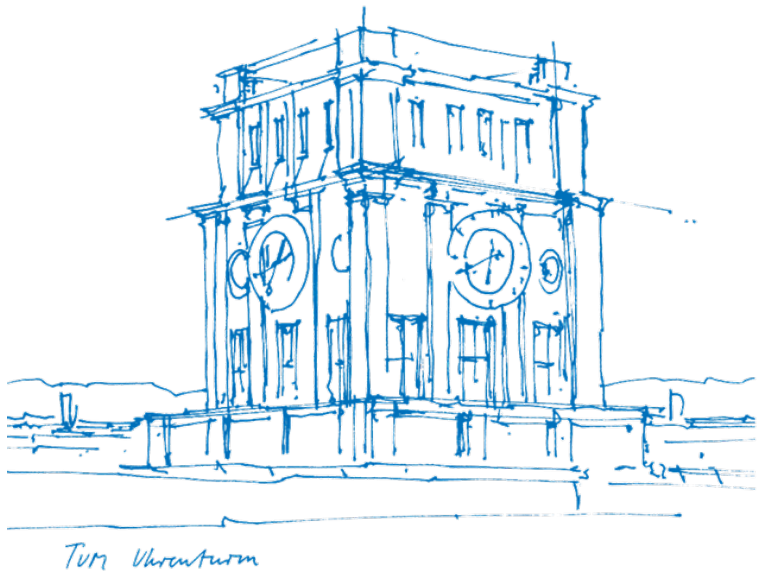
HPC Lab

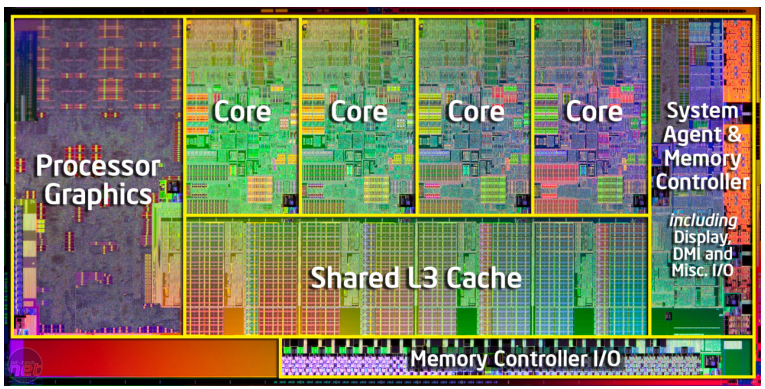
Session 3: MPI

Carsten Uphoff, Chaulio Ferreira, Michael Bader
TUM – SCCS

20 November 2017



MPI: From Kindergarten to Large Scale



Sandy Bridge CPU, `http://www.bit-tech.net/hardware/cpus/2011/01/03/intel-sandy-bridge-review/1`

SuperMUC, `https://www.lrz.de/presse/fotos/`

Hello World

```c
#include <mpi.h>
void main(int argc, char **argv) {
  int rank, size;
  MPI_Init(&argc, &argv);
  MPI_Comm_rank(MPI_COMM_WORLD, &rank);
  MPI_Comm_size(MPI_COMM_WORLD, &size);

  printf("Hello World! (rank %d of %d)", rank, size);

  MPI_Finalize();
}
```

- Compile:
  `mpicc -o hello hello.c`
- Execute:
  `mpiexec -n number_of_processes ./hello`

Hello World

```
1  #include <mpi.h>
2  void main(int argc, char **argv) {
3    int rank, size;
4    MPI_Init(&argc, &argv);
5    MPI_Comm_rank(MPI_COMM_WORLD, &rank);
6    MPI_Comm_size(MPI_COMM_WORLD, &size);
7
8    printf("Hello World! (rank %d of %d)", rank, size);
9
10   MPI_Finalize();
11 }
```

- `int MPI_Comm_size(MPI_Comm comm, int *size)`
  Returns the number of processes in the communicator
- `MPI_COMM_WORLD`
  Predefined standard communicator; includes all processes of a parallel application.
- `int MPI_Comm_rank(MPI_Comm comm, int *size)`
  Returns the process number of the executing process.

Point-to-Point

```
1 MPI_Send(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
      communicator);
2 MPI_Recv(void *buf, int count, MPI_Datatype datatype, int source, int tag, MPI_Comm
      communicator, MPI_Status *status);
```

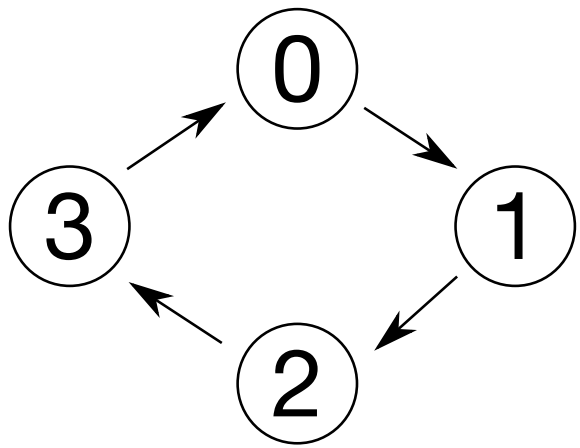
- Blocking operations (return when buffer can be reused)
- *rank* (dest/source) and tag of send- and recieve-call must match
- Wildcards for receive-calls:
  `MPI_ANY_SOURCE`, `MPI_ANY_TAG`, `MPI_STATUS_IGNORE`
- Messages with the same source and destination rank (and the same tag) do not overtake each other (order preservation)

Datatypes

MPI	C
MPI_CHAR	signed char
MPI_SHORT	signed short int
MPI_INT	signed int
MPI_LONG	signed long int
MPI_UNSIGNED_CHAR	unsigned char
MPI_UNSIGNED	unsigned int
⋮	⋮
MPI_FLOAT	float
MPI_DOUBLE	double

Deadlock

```
1 //...
2 int rank, size, dest, src;
3 double *s_buf, *r_buf;
4 //...
5 dest = (rank + 1) % size;
6 src = (rank - 1 + size) % size;
7 MPI_Send(s_buf, 2, MPI_DOUBLE, dest, 0, MPI_COMM_WORLD);
8 MPI_Recv(r_buf, 2, MPI_DOUBLE, src, 0, MPI_COMM_WORLD, MPI_STATUS_IGNORE);
9 //...
```



Non-Blocking

```
1 MPI_Isend(void *buf, int count, MPI_Datatype datatype, int dest, int tag, MPI_Comm
      communicator, MPI_Request *request);
2 MPI_Irecv( void *buf, int count, MPI_Datatype datatype, int source, int tag,
      MPI_Comm communicator, MPI_Request *request);
```

- Returns immediately
- Separates communication into three phases:
  - Initiate communication
  - Do something else
  - Wait for communication to finish
- MPI_Request-object is used to test / wait for completition.

Non-Blocking

```
1 MPI_Wait(MPI_Request *request, MPI_Status *status);
```

Waits until pending communication is finished

```
1 MPI_Test(MPI_Request *request, int *flag, MPI_Status *status);
```

Tests if pending communication is finished

```
1 MPI_Waitall, MPI_Testall, MPI_Waitany, MPI_Testany, MPI_Waitsome, MPI_Testsome
```

There's more...

Collective Operations

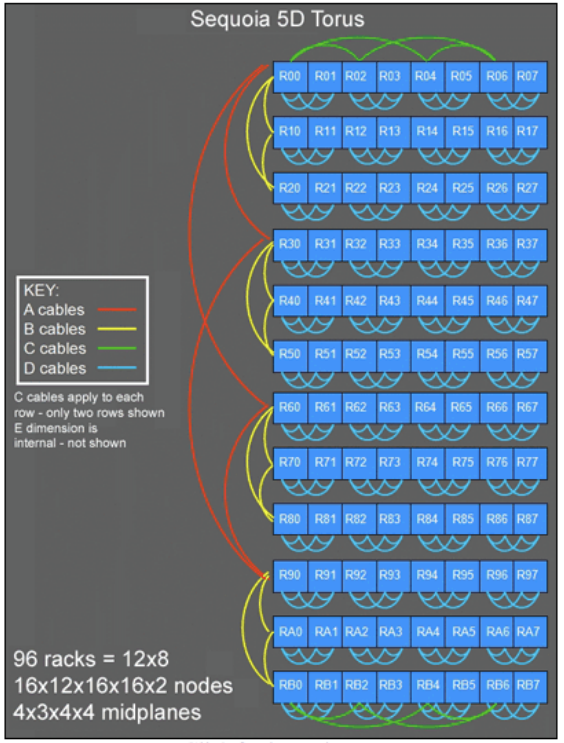
- Three types of collective operations
  - Synchronization (`MPI_Barrier`, ...)
  - Communication (`MPI_Bcast`, ...)
  - Reduction (`MPI_Allreduce`, ...)
- Must be executed by all processes of the communicator
- In MPI-2 all collective operations are blocking operations since MPI-3: Non-blocking collective operations available

Network: BG/Q

computing.llnl.gov

BG/Q Networks

- 5D Torus:
  - Interconnects all compute nodes; every compute node is connected to each of its ten nearest neighbors.
  - Integrates MPI point-to-point, collective and barrier operations into a single network. BG/P and BG/L used three different networks.
  - Peak Bandwidth: 40 GB/s per node (10 links * 2 GB/s * 2-way/bidirectional)
  - Measured Performance - using IBM's low-level, optimized messaging interface (PAMI) software:
    - All-to-all: 97% of peak
    - Bisection: > 93% of peak
    - Nearest-neighbor: 98% of peak
    - Collective: Floating point reductions at 94.6% of peak
  - Hardware Latency: 80 ns nearest neighbor; 3 us worst case (96 rack system)
  - DMA (Direct Memory Access) engine offloads work of injecting and receiving packets from the core - leaves more cycles for the core to do computations
  - Electrical signaling within a midplane; optical links between midplanes through link chips that convert electrical to optical.
  - Hardware assisted barriers and floating point support for collectives in network
  - Single pass floating point reductions at near link bandwidth; bit reproducible
  - The torus also includes an extra 11th link - the I/O link:
    - Used to ship I/O from compute nodes to I/O nodes
    - To match I/O bandwidth to the external file system, only some compute nodes have the I/O link attached to an I/O node



https://computing.llnl.gov/tutorials/bgq/

Topologies

- Processes of a communicator (e.g. `MPI_COMM_WORLD`) can be mapped to a
  - cartesian topology
  - graph topology
- Allow convenient process naming with cartesian process coordinates
- May lead to better performance (network aware programming)

Cartesian

```
1 MPI_Cart_create(MPI_Comm comm_old, int ndims, int *dims, int *periods,
2     int reorder, MPI_Comm *comm cart)
```

Creates a communicator with cartesian topology

```
1 MPI_Cart_sub(MPI_Comm comm, int *remain dims, MPI_Comm *newcomm)
```

Cuts a grid up into “slices”

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

MPI, B. Barney, LLNL, https://computing.llnl.gov/tutorials/mpi/#Virtual_Topologies

Cartesian

```
1 MPI_Cart_rank(MPI_Comm comm, int *coords, int *rank)
```

Converts grid coordinates into process rank

```
1 MPI_Cart_coords(MPI_Comm comm, int rank, int maxdims, int *coords)
```

Returns the grid coordinates of process rank

```
1   MPI_Cart_shift(MPI_Comm comm, int dir, int disp, int *rank_source, int *rank_dest)
```

Returns neighbor ranks w.r.t. direction and displacement

0 (0,0)	1 (0,1)	2 (0,2)	3 (0,3)
4 (1,0)	5 (1,1)	6 (1,2)	7 (1,3)
8 (2,0)	9 (2,1)	10 (2,2)	11 (2,3)
12 (3,0)	13 (3,1)	14 (3,2)	15 (3,3)

MPI, B. Barney, LLNL, `https://computing.llnl.gov/tutorials/mpi/#Virtual_Topologies`

Systems of Linear Equations

$$Ax = b$$

- A regular, b known
- Direct methods: Gauß, LU-decomposition, QR-decomposition
- Iterative methods: Splitting methods (Jacobi, Gauß-Seidl, SOR), projection methods (CG, GMRES, BiCGSTAB)

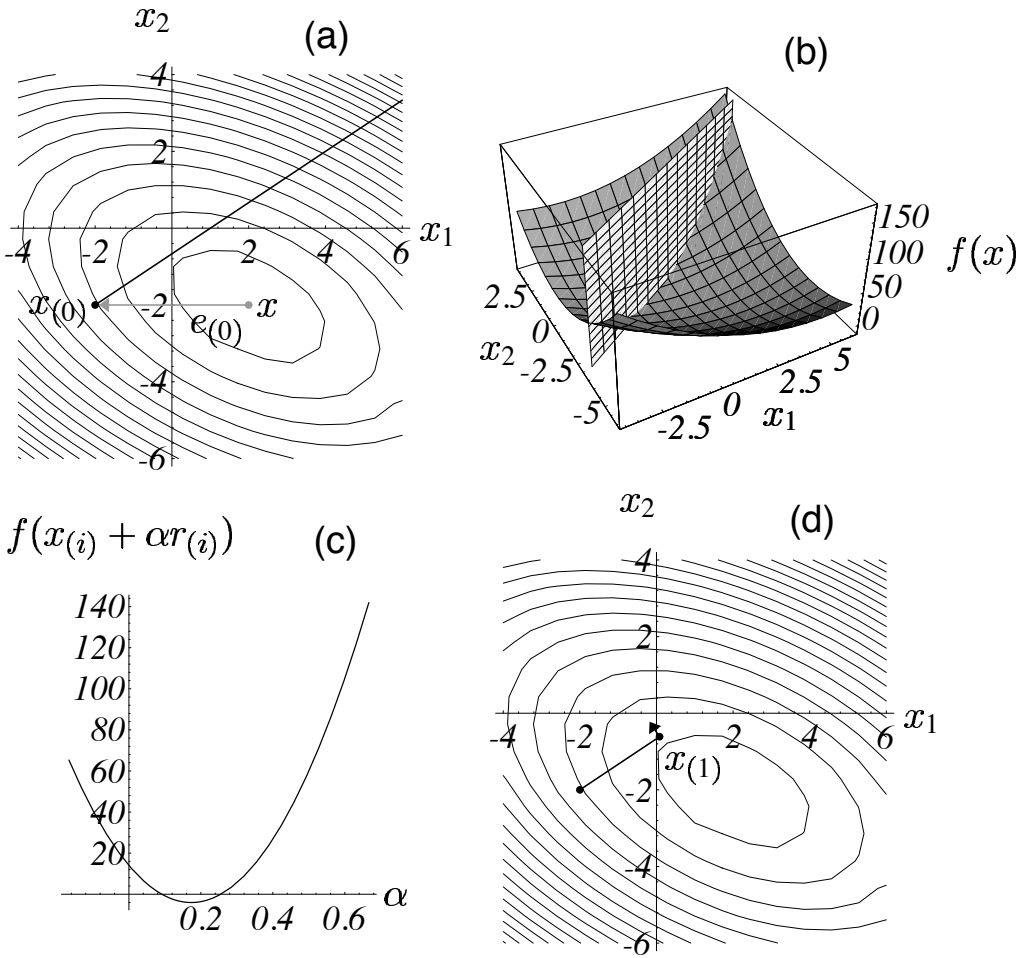
Conjugate Gradients Method

$$Ax = b$$

Solve SLEs with symmetric and positive definite matrices, equivalent to minimizing:

$$g(x) = \frac{1}{2}x^T Ax - b^T x + c$$

Algorithm



$i \Leftarrow 0$
$r \Leftarrow b - Ax$
$\delta \Leftarrow r^T r$
$\delta_0 \Leftarrow \delta$
While $i < i_{max}$ and $\delta > \varepsilon^2 \delta_0$ do
    $q \Leftarrow Ar$
    $\alpha \Leftarrow \frac{\delta}{r^T q}$
    $x \Leftarrow x + \alpha r$
    If i is divisible by 50
        $r \Leftarrow b - Ax$
    else
        $r \Leftarrow r - \alpha q$
    $\delta \Leftarrow r^T r$
    $i \Leftarrow i + 1$

J. R. Shewchuk, An Introduction to the Conjugate Gradient Method Without the Agonizing Pain,
`http://www.cs.cmu.edu/~quake-papers/painless-conjugate-gradient.pdf`

Laplace's equation

$$\Delta f(x) = 0$$

- $x \in \mathbb{R}^2$
- Dirichlet boundary conditions
- regular full grid
- no need to assemble the matrix: Use an implicit given operator
- finite differences; in 1D: $f''(x) = \frac{f(x+h)-2f(x)+f(x-h)}{h^2}$