Overview of the MPI Standard and Implementations

Hauptseminar “Cluster Computing”
21.05.2004

Christina Zeeh
Fakultät Informatik, Elektrotechnik und Informationstechnik
Universität Stuttgart

Overview

- MPI = Message Passing Interface
- Topics
  - Message Passing
  - MPI standard
  - MPI implementations
  - Future trends

Multiprocessors vs. Multicomputers

	Multiprocessors	Multicomputers
Memory	shared memory	distributed memory
Communication	data in shared memory	message-passing
Synchronization	explicit	implicit

Properties of Parallel Programs

- Distribution of work (and associated data, if memory is distributed) – decomposition
- Communication of data and results
- Synchronization, for example when one computation needs to wait for another computation's results

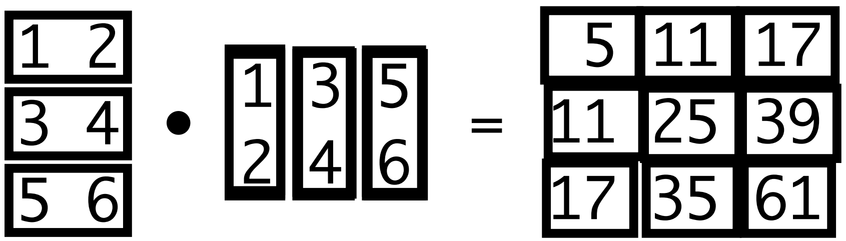
Decomposition Examples

```
for (i=0;i++;i<1000) {
      a[i] = a[i]-b[i];
}
```

```
for (i=0;i++;i<1000) {
      c[i] = c[i] + a[999-i];
}
```

synchronization,
deadlocks!

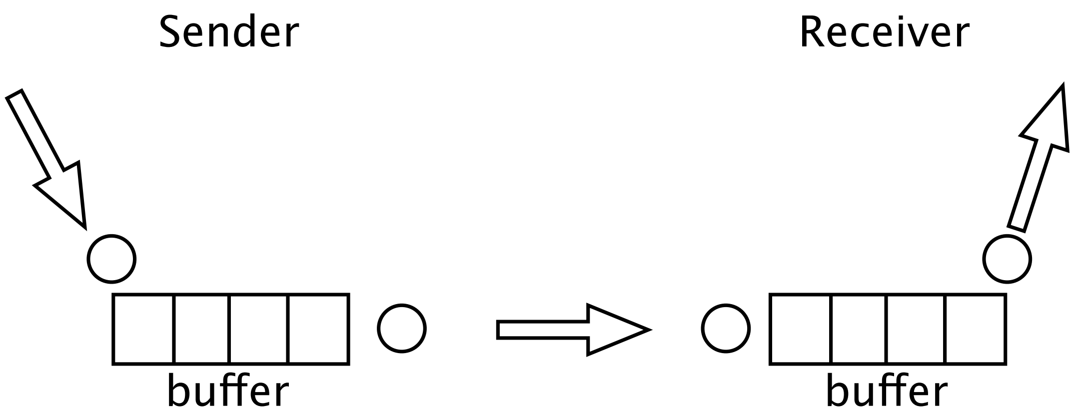
Process 0: i=0..249
Process 1: i=250..499
Process 2: i=500..749
Process 3: i=750..999



Message Passing

- Communication through messages
- Two primitives: `send` and `receive`
- Messages imply synchronization
- Blocking, buffering and reliable communication introduce different message passing semantics

Example: Blocking Send



Sender blocks until message is ...

- copied to sender's buffer (if buffer is full)
- sent
- received at the receiver
- delivered at the receiver

Characteristics

- Low-level approach
  - Work must be explicitly distributed
  - Data must be explicitly distributed
- Interactions require both sides to actively participate
- Danger of deadlocks

→ Writing message passing programs is considered to be “hard”

Alternatives

- Shared memory
  - POSIX threads
  - OpenMP (user specifies work distribution, no data distribution)
- Distributed memory
  - Virtual shared memory

Before MPI ...

- ... message-passing was already an established paradigm in the early 90s
- Some consensus had been reached on what a message passing interface needs to provide ...
- ... but most available message passing libraries (PVM, Express, P4, Intel NX/2, ...) were mutually incompatible

  $\Rightarrow$ developing portable applications was difficult

MPI Standard

- Developed by the MPI Forum:
  - initial meeting at the Workshop on Standards for Message Passing in a Distributed Memory Environment in April 1992
  - parallel computer vendors and researchers
  - not an official standardization organization
- First version of the standard released in 1994

MPI-1

- History
  - updated version (1.1) released in 1995
  - version 1.2 is part of MPI-2
- Point-to-Point Communication
- Collective Communication
- Communicators, Groups, Contexts
- Datatypes
- Bindings for Fortran-77 and C

Hello, World!

```
char msg[15];
int myrank;
MPI_Status status;
MPI_Init( &argc, &argv );
MPI_Comm_rank( MPI_COMM_WORLD, &myrank );
if (myrank == 0)    /* process 0 */
{
    strcpy(msg,"Hello World");
    MPI_Send(message, strlen(msg), MPI_CHAR, 1, 99, MPI_COMM_WORLD);
}
else                /* process 1 */
{
    MPI_Recv(msg, 15, MPI_CHAR, 0, 99, MPI_COMM_WORLD, &status);
    printf("Message: %s\n", msg);
}
MPI_Finalize();
```

Compiling and Running an MPI Program

- MPI is simply a library ⇒ use standard compiler
- Most MPI implementations provide a command mpirun to run MPI programs
- MPI-2 standard specifies `mpiexec`
- On multiprocessors: tell `mpirun` how many processors to run on
- On multicomputers: MPI implementations usually provide a way to start the program on multiple nodes (remote login, daemons on each node, ...) using `mpirun`

Rank and Communicator

- Communicator
  - Each process belongs to one or more communicators
  - Communicators provide a way to define the scope of a communication and manage processes
  - All processes of one MPI program belong to `MPI_COMM_WORLD`
- Rank
  - Processes in a communicator are numbered sequentially, rank is comparable to a process ID

MPI Messages

```
MPI_SEND (buf, count, datatype, dest, tag, comm)
```

- buf = buffer containing the data to be sent
- count = number of elements in the buffer
- datatype = datatype of the buffer elements
- dest = destination's rank
- tag = message tag
- comm = communicator

} + source = envelope

Datatypes

- Basic datatypes:
  - MPI_CHAR
  - MPI_INT
  - MPI_FLOAT
  - MPI_BYTE (uninterpreted)
  - MPI_PACKED (explicit packing/unpacking)
  - ...
- Derived datatypes

Derived Datatypes

- Needed for ...
  - messages with mixed datatypes
  - non-contiguous data
- One could manually pack/unpack the data
  ⇒ overhead (memory-to-memory copies)
- Derived datatypes are constructed from basic datatypes (or other derived datatypes)
- Type map specifies layout of the data structure (datatypes and displacements)
- 4+ constructors: `MPI_TYPE_CONTIGUOUS`, `MPI_TYPE_VECTOR`, `MPI_TYPED_INDEXED`, `MPI_TYPE_STRUCT`

Point-To-Point Communication

- Blocking
- Communication modes:
  - standard
  - buffered
  - synchronous
  - ready
- Non-blocking (”immediate”)
  - initiation and completion (`MPI_TEST/MPI_WAIT`)

Collective Communication

- Involves the members of a communicator
- Provides
  - barrier synchronization
  - broadcast (one-to-all)
  - scatter/gather operations
  - reduction operations (predefined: max, min, sum, ...)

Virtual Topologies

- Many parallel programming problems are multi-dimensional $\Rightarrow$ sequential process naming provided by ranks is inconvenient
- Virtual topology information can be used by MPI runtime to optimize assignment of processes to physical hardware
- Creating a new virtual topology creates a new communicator

Virtual Topologies (cont.)

- MPI provides two kinds of virtual topologies
  - graph topologies – communication with neighboring processes
  - cartesian topologies – communication with neighboring grid points, convenient naming through rank/coordinate mapping

MPI-2

- Published in 1997
- More clarifications on MPI 1.1 (→MPI 1.2)
- New features
  - Bindings for Fortran-90 and C++
  - One-sided communication
  - Dynamic processes
  - Parallel I/O

One-Sided Communication

- Remote memory access (RMA)
- Primitives MPI_PUT and MPI_GET (⇒ explicit data transfer ≠ shared memory)
- “Windows” make memory available for remote access
- Need explicit synchronization!

One-Sided Communication Synchronization

- Active target communication
  - MPI_WIN_FENCE
  - MPI_WIN_START, MPI_WIN_COMPLETE, MPI_WIN_POST, MPI_WIN_WAIT
- Passive target communication
  - MPI_WIN_LOCK, MPI_WIN_UNLOCK

Dynamic Processes

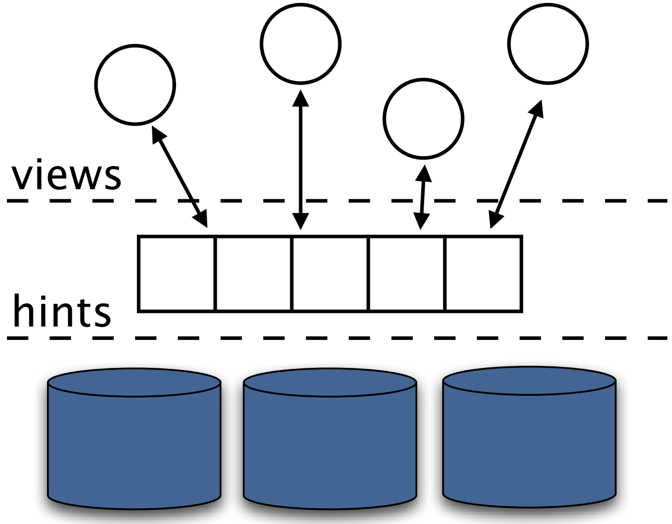
- MPI-2 allows a running application to ...
  - create processes
  - terminate processes
  - establish communication
- Underlying process management system is being used
- “info” argument allows for environment-specific functionality (but compromises portability)

Dynamic Processes (cont.)

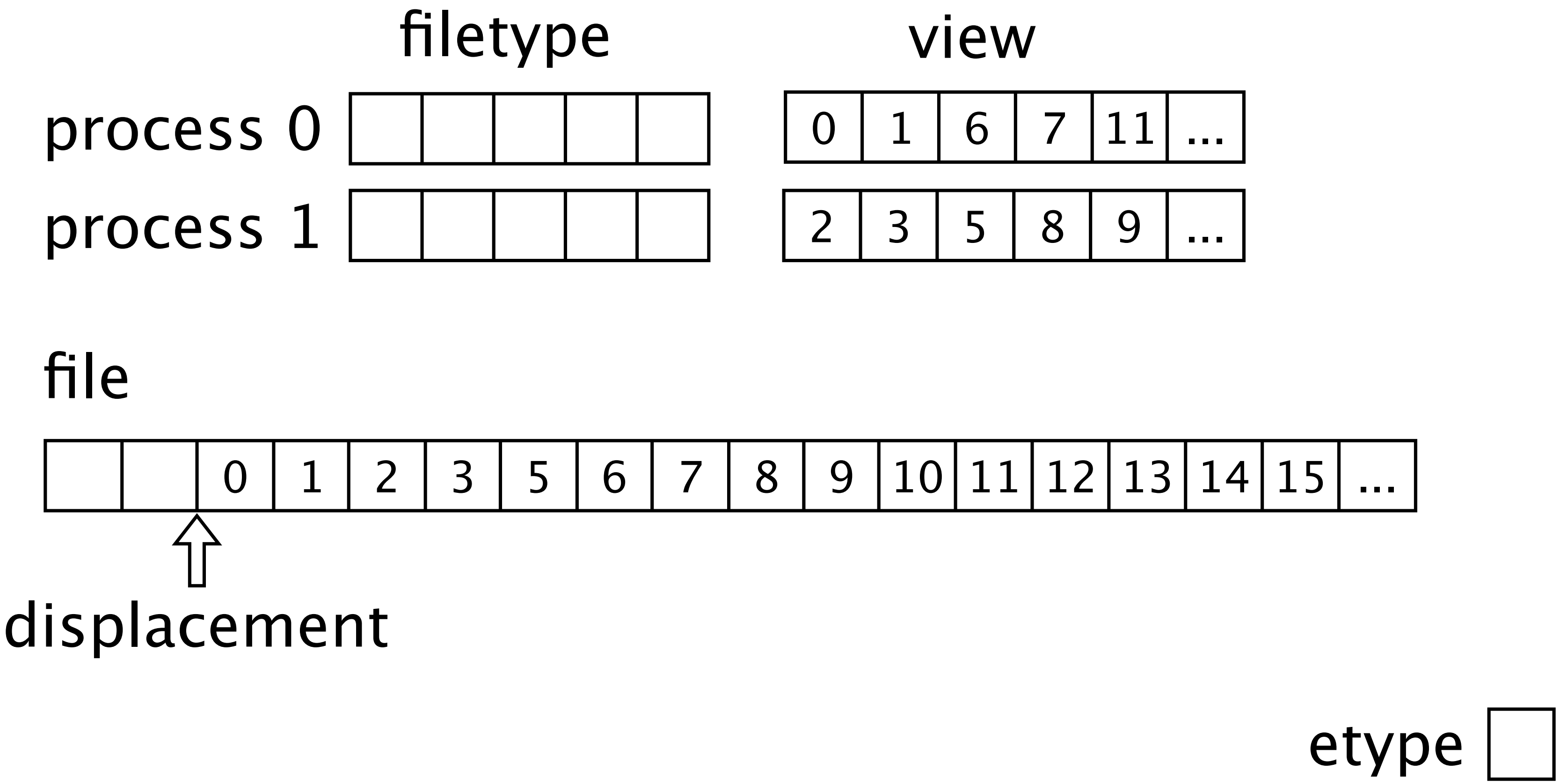
- Process creation functions return an intercommunicator for the new process(es)
- New processes can acquire an intercommunicator with `MPI_COMM_GET_PARENT`
- MPI-2 also provides facilities for establishing communication to non-related processes

Parallel I/O

- In parallel applications ...
  - files are accessed concurrently
  - data in one file may be shared by many processes, they need to read/write non-contiguous pieces
  - file access needs to be coordinated



Parallel I/O (cont.)



Parallel I/O (cont.)

- Positioning
  - explicit (offset)
  - implicit (file pointer, individual or shared)
- Synchronism
  - blocking
  - non-blocking and split collective
- Coordination
  - collective
  - non-collective

Ideas that didn't make it into MPI-2

- Support for real-time processing (whole chapter)
  → MPI/RT standard (currently version 1.1)
- Starting processes dynamically without establishing communication (independent processes)
- Two-phase (split) collective operations
- ...

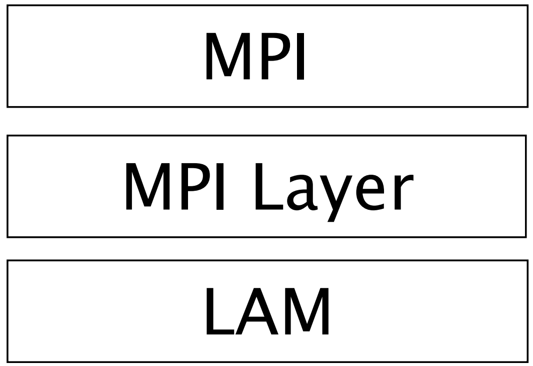
MPI Implementations

- Vendor-supplied: Sun, SGI, HP, NEC, ...
- Free/Open source: MPICH, LAM/MPI, ...
- Commercial: ChaMPIon/Pro, WMPI, ...
- All of them implement MPI-1 (or most of it)
- Most MPI-2 implementations are still incomplete

LAM/MPI

- Implements MPI-1 and large portions of MPI-2
- Originally developed at the Ohio Supercomputing Center, now at the University of Notre Dame.
- User-level daemon for process management etc. ⇒ the actual startup using `mpirun` is fast
- Supports most POSIX platforms

LAM/MPI Architecture

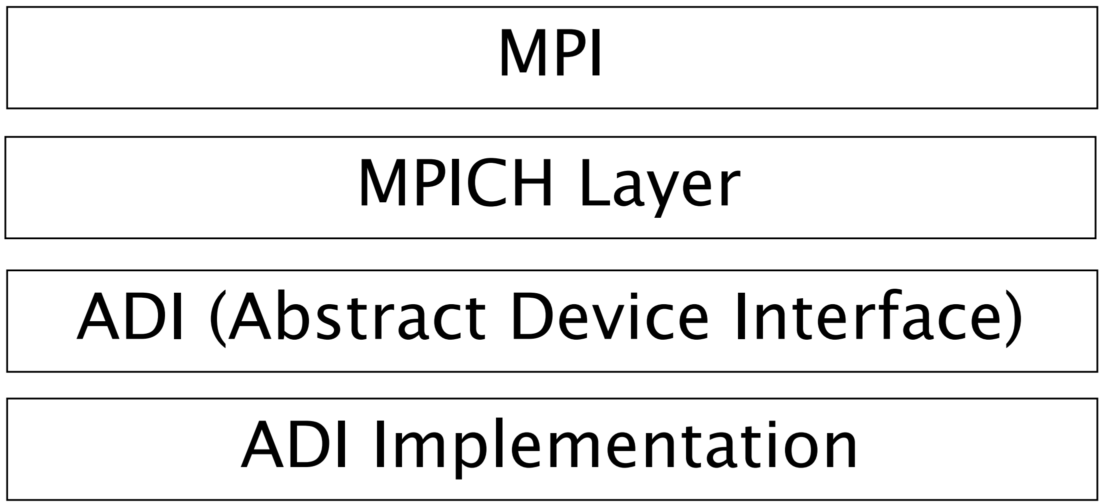


- LAM runtime environment
- System Services Interface (SSI) provides support for modules, currently specified:
  - boot – replacement for LAM daemons
  - coll – collective operations
  - cr – checkpoint/restart
  - rpi – low-level point-to-point communication

MPICH

- Implements all of MPI-1 and some features of MPI-2 (most significant: MPI-IO using ROMIO)
- Developed alongside the standard
- Supports most Unix flavors and Windows NT
- Architecture provides for excellent portability ⇒ numerous successful MPICH-derivates
  - MPICH-V (fault tolerance)
  - MP-MPICH (heterogeneous clusters)
  - MVAPICH (using the native Verbs Level Interface (VAPI) of InfiniBand)

MPICH Architecture



MPICH-G2

- Grid-enabled MPI implementation based on MPICH (ADI implementation ⇒ globus2 device)
- Issues:
  - different hardware and software platforms
  - diverse network conditions
  - cross-site authentication
- Uses the Globus Toolkit
  - Globus Resource Allocation Manager (GRAM)
  - Grid Security Infrastructure (GSI)
  - Monitoring and Discovery Service (MDS)
  - Global Access to Secondary Storage (GASS)

MPICH-G2 (cont.)

- Chooses most efficient communication method
  - vendor-supplied MPI (vMPI)
  - Globus communication for TCP
- Collective operations are aware of the actual topology of the grid
- Topology information is stored as communicator attributes and can thus be used by an application

MPICH2

- Implements all of MPI-1 and some of MPI-2 (complete implementation in progress)
- Re-Implementation of MPICH, all new MPICH development focuses on MPICH2
- Current status: Beta test version available
  - MPI-2: all of MPI-IO, preliminary one-sided communication
  - Limited device support (TCP sockets, shared memory)

ChaMPIon/Pro

- Full MPI-2 implementation
- Supports RedHat on IA32, SuSE on AMD Opteron (both including InfiniBand, Myrinet), others planned
- Commercial implementation: 300-400$ per CPU + 20% p.a. support and maintenance (minimum 1 year)
- Also offer an MPI-1-only implementation (MPI/Pro) with extensive OS support (Linux, Windows, OS X)

Sun MPI

- Implements MPI-1 and almost all of MPI-2
- Extensive software environment available
  - Sun Parallel File System
  - Prism (debugger and performance analyzer)
  - S3L (Scalable Scientific Subroutine Library)
  - Sun Cluster Runtime Environment
  - Cluster Console Manager
- Heavily tuned
  - uses shared memory, when possible
  - important functions have been optimized
  - tuned collective operations
  - environment variables for fine-tuning
- Supports Sun platform (Solaris 8/9)

Comparison

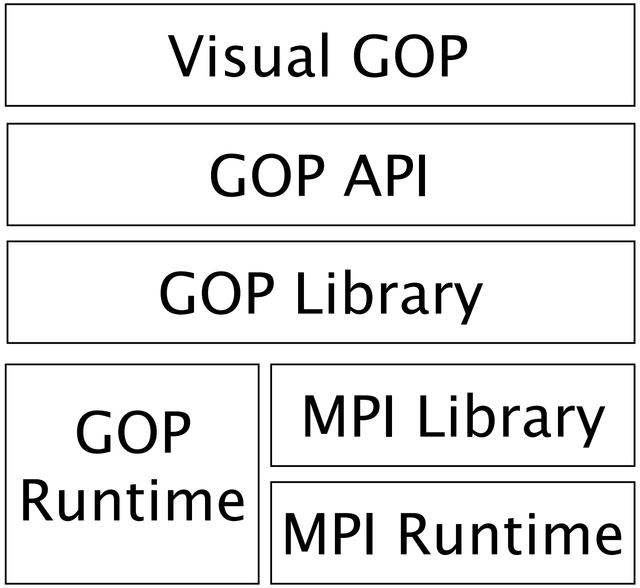
	MPICH	LAM/MPI	Sun MPI	ChaMPIon/ Pro
MPI-1	✔	✔	✔	✔
MPI-2	•	•	•	✔
- one-sided comm.	✘	•	•	✔
- parallel I/O	•	✔	✔	✔
- dynamic processes	✘	✔	•	✔
- C++ / Fortran 90	•/•	✔/✘	✔/✔	✔/✔
IMPI Support	✘	✔	✘	✘
Grid Capabilities	✔	✔*	✔	✘
Debugging Facilites	✔	✔	✔	✔

* beta

Graph-Oriented Programming (GOP)

- GOP provides a high-level abstraction for MPI
- A GOP program consists of
  - graph construct describes the logical relationships between local programs
  - local programs (LPs)
  - LP-to-node mappings
  - node-to-processor mappings (optional)

Graph-Oriented Programming (GOP)



- GOP API
  - graph-oriented point-to-point, collective, synchronization and query primitives
  - enhanced communication support (node group, graph topology)
- GOP runtime provides graph management (updates, query, synchronization etc.)

MPI – Future Trends

- MPI-3? Nowhere in sight
- MPI in heterogeneous environments
  - Interoperable MPI (IMPI) - run MPI applications across multiple implementations
- MPI on grids
- Faster networking hardware
  - Exploiting all of the features the InfiniBand architecture provides
- Scalability
- Tool support

Questions?