Improving the Dynamic Creation of Processes in MPI-2

Márcio C. Cera, Guilherme P. Pazzi, Elton Mathias, Nicolas Maillard and Philippe O. A. Naveaux

Work sponsored by CNPq, CAPES and HP Brazil

MPI, MPI-2, ...

- Message Passing Interface is the de-facto standard for Cluster Computing
  - Inherited from PVM
  - MPI-2 does not provide the dynamic creation/management of processes
- MPI-2 has been defined in 1998.
  - Portable, on Windows, etc.
  - Dynamic creation of processes (MPI_Comm_spawn)
- Recent implementations of MPI-2:
  - LAM since the start of the 2000
  - LAM/PTO (Jan. 2002)
  - LAM/PTO (Aug. 2002)
- Towards a MPI for Grids?
  - MP-G4: MP-Grid Application supports heterogeneity, but not the dynamicity.
  - Developing a MONET-MPI (G4-MONET) and GLAM (SLAMS)
  - Goals of MONET-MPI 1.0
    - Open MPI, fork between MP-FT and MPI
    - Fully distributed

MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);

  - cmd: name of the MPI executable.
  - argv, argc: command line arguments to be passed to 'cmd'
  - nbprocs: number of MPI processes to be created.

MPI_Comm_spawn()

- MPI_Comm_spawn(cmd, argv, argc, nbprocs, info, root, comm_root, &intercomm, err);

  - info: backdoor left to the implementation.
  - MPI-2 defines the datatype 'MPI_Info'
  - Ex. of use:

    MPI_Info_set(info, "tam_spawn_sched_round_robin", rank)

    - Starts a Round-Robin from proc number 'rank'
    - (Round-Robin is the default)
Communication between the Processes

- The parent uses the inter-communicator to send/recv messages with its children.
- The children have to call MPI_Get_parent() to obtain their parent's communicator.
  - If the return is NULL, the children have been "mipruned" directly, and not MPI_Comm_spawned.
  - The parent has rank 0 in this communicator.

Example: Fibonacci with MPI-2

```c
if (rank < 2) {
    mpi_num = 0, mpi_comm, p, parent, rank;
}
else
    mpi_num, mpi_comm, p, local_num, rank, mpi_comm_par, children_num, children_num;

mpi_num = 0; mpi_comm, p, local_num, rank, mpi_comm_par, children_num;

mpi_start = 0, mpi_end, mpi_comm, mpi_comm_par, children_num, mpi_comm_par, children_num;

mpi_end = 0, mpi_comm, mpi_comm_par, children_num, mpi_comm_par, children_num;

MPI_Finalize(0);
```

Two Main Issues with Dynamic Processes

- How to be efficient in the communication between parent and children?
  - If anybody wants to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
  - One should hierarchize the processes
    * → Divide & Conquer.

- How does MPI_Comm_spawn allocate the processes?
  - Default: Round-Robin from a fixed rank (0).
  - Problem if a series of Spawns are repeated.
  - Problem when more than one process perform spawns in parallel...

Two Main Issues with Dynamic Processes

- How to be efficient in the communication between parent and children?
  - If anybody wants to communicate with everybody, the comm have to be merged (MPI_Comm_merge).
  - One should hierarchize the processes
    * → Divide & Conquer.

- How does MPI_Comm_spawn allocate the processes?
  - Default: Round-Robin from a fixed rank (0).
  - Problem if a series of Spawns are repeated.
  - Problem when more than one process perform spawns in parallel...
Two Main Issues with Dynamic Processes

- How to be efficient in the communication between parent and children?
  - If anybody wants to communicate with everybody, the comm have to be merged (MPI Comm: merge).
  - One should hierarchize the processes
    - Grid & Compact
- How does MPI_Comm_spawn allocate the processes?
  - Default: Round-Robin from a fixed rank (0).
  - Problem if a series of Spawns are repeated.
  - Problem when more than one process perform spawes in parallel...

Native Allocation of Processes

- The native mechanism may allocate all processes to one processor!

<table>
<thead>
<tr>
<th>Environment</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>20 spawns of 2 processes</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1 process on 20 processes</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>4</td>
</tr>
</tbody>
</table>

- Improvement with one variable that controls where to launch the processes.

<table>
<thead>
<tr>
<th>Environment</th>
<th>Node 1</th>
<th>Node 2</th>
<th>Node 3</th>
<th>Node 4</th>
<th>Node 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Different LANs, second selection</td>
<td>30</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>with balancing schedule</td>
<td>3</td>
<td>4</td>
<td>8</td>
<td>2</td>
<td>3</td>
</tr>
</tbody>
</table>

Solution: a Centralized Scheduler

- Simple idea:
  - A daemon is run together with the MPI application to centralize the allocation decision.
  - MPI_Comm_spawn at MPI_Finalize() are realigned to notify the daemon of process creation/termination.

- The scheduler daemon:
  - Can manage the task graph of the application.
  - Can decide about the allocation of the spawned processes, with a Round-Robin algorithm.
  - Centralized if R.
  - Can monitor the load and base the decision about the load of each node.
  - Etc...

- Simple tests have been performed with a prototype
  - To be included in a LAM distribution!

Implementation of the Scheduler

1. MPI_Comm_spawn' Notification of the creation of a process
2. Scheduling decision
3. Physical creation
4. Notification of the completion of the process

Three Experiments

1. Application of the centralized RR to the computation of Fib(7), Fib(10) and Fib(13).
   - Dynamic creation of processes and resources
   - Load balancing with dynamic resources.

2. Recursive computation of the prime numbers in the interval [1 .. N], with measure of the load
   - Irregular run-time
   - Improving the computation time.

3. Round-Robin with a dynamically increasing number of nodes (larget)
   - Dynamic creation of processes and resources
   - Load balancing with dynamic resources.

1 - Fibonacci – Native Solution vs. Centralized Round-Robin Allocation

<table>
<thead>
<tr>
<th>Number of processes on each node</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nation solution</td>
</tr>
<tr>
<td>Centralized RR solution</td>
</tr>
</tbody>
</table>
1 - Fibonacci – Native Solution vs. Centralized Round-Robin Allocation

2 - Prime Numbers – Balancing the Load of an Irregular Application

3 - Fibonacci – Allocation of Processes with lamgrow

4 - Fibonacci – Allocation of Processes with lamgrow
3 - Fibonacci – Allocation of Processes with lamgrow

Limitations & Next Steps

- Limited to LAM-MPI
  - Yet, easy to port!
    - The only LAM-dependent part is the integration into the Mpi_Cores' spares implementation.

- Lamgrow is fine... What about lamthink?
  - One needs some checkpoint/replica mechanism...
    - Open-MPI could provide it?

- In a view to working with coarse-grained applications, the benchmarks are somewhat limited...
  - Current work includes "real-world" applications.

- Using such mechanisms in Grids?
  - Does MPI2 run on the Grid?
    - Globus enables MPI distribution does not seem to focus MPI 2...

Conclusions

- Dynamic creation of processes with MPI-2 is okay.
  - Interesting for coarse-grained applications
  - One needs to find a way to manage efficiently the communication
    - Parent/Child
    - LAM enables the dynamic integration of new resources (lamgrow)

- LAM’s native allocation of Spawned processes is weak.
  - Well, it respects the norm!
    - A simple, centralized solution leads to clear improvements.
    - Why not providing such add-ons in the distributions?

- Natural idea: distribute the scheduler
  - Workstealing?

Any return will be welcome!

nicolas@inf.ufrgs.br