Hybrid Bulk Synchronous Parallelism Library for Clustered SMP Architectures

Khaled Hamidouche, Joel Falcou, Daniel Etiemble
hamidou, joel.falcou, de@lri.fr
LRI, Université Paris Sud 11
91405 Orsay, France
Outline

- Introduction
- BSP model
- BSP++ library
- Hybrid programming support
- Experimental results
- Conclusion & future works
Introduction

• Today’s machines are hierarchical
  Cluster, SMP, Multi-cores

• Hard to efficiently program
  low level programming model MPI, OpenMP

• Performance depends on
  Application: data size, comm/comp pattern
  Architecture: CPU, bandwidth, …
High level parallel programming tools

- High level parallel programming models
- High performance
- Easy to manipulate
BSP Model (Leslie G Valiant:1990)

• Three components:
  - Machine Model
  - Programming Model
  - Cost model
BSP Model (Leslie G Valiant:1990)

1- Machine Model

- Describes a parallel machine
  - Set of Processors
  - Point to point communication
  - Synchronization

- Experimental Parameters
  
P: Number of processors
  
r: CPU speed (FLOPS)
  
g: Communication speed (sec/byte)
  
L: Synchronization time (sec)
BSP Model (Leslie G Valiant:1990)

2- Programming Model

- Describes the structure as a sequence of steps
BSP Model (Leslie G Valiant: 1990)

3- Cost Model

- Estimates the time

\[ T = \sum \delta_i \]

\[ \delta = W_{\text{max}} + \text{max h.g} + L \]
BSP++

- Object-oriented implementation of the BSML Library \[^{[gava:09]}\] in C++

- Notion of Parallel vector

- Functional programming support
  Boost.Phoenix and C++ lambda-function
BSP++ API

- `par<T>`: Concept of parallel vector, many constructors

- `sync ()`: Explicit synchronization, MPI or OpenMP barrier

- `proj`: `result_of::proj<T> proj (par<T> &)`

- `put`: `result_of:: put<function<T(int)> > put (par<function<T(int)> >& )`
BSP++ API

- **proj**: MPI\_allgather and asynchronous OpenMP copy + sync ()

- **put**: Matrix P: $P_{ij} =$ value of Proc $i$ to send to Proc $j$
  MPI\_alltoall and asynchronous OpenMP copy + sync ().
Example

Inner product program

P0

V

A local computation

r1

r1 r2 r3

An accumulate of the partial results

r

P1

V

A local computation

r2

r1 r2 r3

An accumulate of the partial results

r

P3

V

A local computation

r3

r1 r2 r3

An accumulate of the partial results

r

HLPP2010, Baltimore

K. Hamidouche

Sept, 25, 2010

12/29
Example

BSP++ Inner product

```cpp
#include<bsppp/bsppp.hpp>

int main (int argc, char** argv)
{
    BSP_SECTION(argc, argv)
    {
        par<vector<double> > v;
        par<double > r;

        // step 1 : perform local inner-product
        *r=std::inner_product( v->begin(), v->end(), v->begin(), 0.);

        // the global exchange
        result_of::proj<double> exch = proj (r);

        // step 2 : accumulate the partial results
        *r= std::accumulate (exch.begin(), exch.end() );

        sync ();
    }
}
```
Hybrid programming support

Objection of BSP is the **cost** of **L**
(dominant for large parallel machines)

Table. Variation of L (in ms) and g (in second per M b) on A 4x4 cores machine (AMD machine)

<table>
<thead>
<tr>
<th></th>
<th>MPI</th>
<th>OpenMP</th>
</tr>
</thead>
<tbody>
<tr>
<td>P</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td></td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>g</td>
<td>0.087</td>
<td>0.025</td>
</tr>
<tr>
<td></td>
<td>0.22</td>
<td>0.069</td>
</tr>
<tr>
<td></td>
<td>1.69</td>
<td>0.68</td>
</tr>
<tr>
<td>L</td>
<td>4.46</td>
<td>2.94</td>
</tr>
<tr>
<td></td>
<td>20.8</td>
<td>8.13</td>
</tr>
<tr>
<td></td>
<td>108.0</td>
<td>13.1</td>
</tr>
</tbody>
</table>

**Impact** of OpenMP: synchronization is up to 8 times faster

**Turn** the hybrid BSP machine into two BSP machines with different values of L and g
Hybrid BSP with BSP++

- Same code for both MPI and OpenMP
- Add a split function

\[ \delta = W_{\text{max}} + h_{\text{mpi}} \cdot g_{\text{mpi}} + h_{\text{omp}} \cdot g_{\text{omp}} + L_{\text{mpi}} + L_{\text{omp}} \]
double **omp_inner_prod** (vector<double> const & in, int argc, char ** argv )
{
    double value;
    BSP_SECTION(argc, argv)
    {
        par<vector<double> > v= split (in);
        par<double> r;
        *r = std::inner_product(v->begin(), v->end(), v->begin(), 0.);
        result_of::proj exch = proj(r);
        value = std::accumulate (exch.begin(), exch.end());
    }
    return value;
}

BSP_SECTION(argc, argv)
{
    par<vector<double> > data;
    par<double> result;
    *result= **omp_inner_prod** (*data, argc, argv);
    result_of::proj<double> exch= proj(result);
    *result= std::accumulate (exch.begin(), exch.end() );
}
Experimental results

• Platforms:

  1- **AMD machine:**
  * 2 GHz Quad processor quad cores (16 cores)
  * 16 Gb of RAM (shared memory)
  * gcc4.3, OpenMP 2.0 and OpenMPI 1.3

  2- **CLUSTER machine:**
  * Grid5000 platform; Bordeaux site
  * 4 nodes, Bi-processor Bi-cores (2.6 GHz)
  * gcc4.3, MPICH2.1.0.6 library
Experimental results

• Protocols:

1- BSP++ vs BSPlib:

* AMD machine
* EDUPACK benchmarks (Inprod, FFT, LU)

2- BSP++: MPI vs OpenMP:

* AMD machine
* Inprod, Matrix-vector Multiplication GMV, Matrix-matrix Multiplication GMM and Text Count function of the google MAP reduce Algorithm Benchmarks

3- BSP++: MPI vs Hybrid:

* Cluster machine
* Same benchmarks
1- BSP++ vs BSPlib

Overall execution time for BSP++ on OPENMP and the BSPlib EDUPACK benchmarks on the AMD machine

- Same performances
- No overhead of the generic template implementation
Execution time of the InProd benchmark on the AMD machine for 64 $10^6$ elements
Execution time of the GMV benchmark on the AMD machine with a 8192 x 8192 matrix
2- BSP++: MPI vs OpenMP

Execution time of the GMM benchmark on the AMD machine with 2048 x 2048 matrices

47%
2- BSP++: MPI vs OpenMP

Execution time of the MAP benchmark on the AMD machine for 150000 words list

75% and 86%
Execution time of the InProd benchmark on the Cluster machine for 64 $10^6$ elements
Execution time of the GMV benchmark on the Cluster machine with a 8192 x 8192 matrix
3- BSP++: MPI vs Hybrid

Execution time of the GMM benchmark on the Cluster machine with 2048 x 2048 matrices

60%
Execution time of the MAP benchmark on the Cluster machine for 150000 words list
Conclusion

- MPI and OpenMP as a native targets
  - Both versions scale
  - No overhead of the C++ implementation

- Simplify the design of Hybrid MPI+OpenMP codes
  - Using the same code

- Support a large number of practical development idioms
Future works

- Implementation of BSP++ on **Cell** and **GPU**: Hybrid MPI+OpenMP+GPU

- BSP based **containers and algorithms**: Write a subset of C++ standard library as BSP algorithm

- A framework for an automatic Hybrid MPI+OpenMP code generation.
Hybrid Bulk Synchronous Parallelism Library for Clustered SMP Architectures

Khaled Hamidouche, Joel Falcou, Daniel Etiemble
hamidou, joel.falcou, de@lri.fr
LRI, Université Paris Sud 11
91405 Orsay, France