Leveraging Non-Blocking Collective Communication in High-Performance Applications

Torsten Hoefler, Peter Gottschling and Andrew Lumsdaine

Open Systems Lab
Indiana University
Bloomington, USA

20th ACM Symposium on Parallelism in Algorithms and Architectures - SPAA'08

Munich, Germany

June, 14th 2008

Introduction

Features of non-blocking collective operations

- hide full communication latency by overlapping
- use the available bandwidth better
- avoid detrimental effects of pseudo-synchronization/process skew
- make efficient use of the new semantics

LibNBC and MPI

- implements all MPI collectives non-blocking
- overhead-optimized implementation
- special InfiniBand™ optimizations
- progress thread

Problems and a Solution

Challenges for the Programer

- rearrange the algorithm to overlap
- implement and debug non-blocking communication
- optimize overlap (e.g., message sizes)

Overcoming the Problems

- semi-automatic approach for applications with independent data
- covers many applications that fit the map-reduce model
- many scientific applications (e.g., parallel data processing, Fourier transformation, parallel sorting, FEM methods, ...)

A typical Program - Parallel Compression

```
my_size = 0;
for (i=0; i < N/P; i++) {
  my_size += compress(i, outptr);
  outptr += my_size;
}
gather(sizes, my_size);
gatherv(outbuf, sizes);
```

Parallel Compression - Overlapping Version

```
for (i=0; i < N/P; i++) {
  my_size = compress(i, outptr);
  gather(sizes, my_size);
  igatherv(outptr, sizes, hndl[i]);
  outptr += my_size;
  if(i>0) waitall(hndl[i-1], 1);
}
waitall(hndl[N/P], 1);
```

```
for (i=0; i < N/P/t; i++) {
  size = 0;
  for (j=i; j < i+t; j++) {
    my_size = compress(i*t+j, outptr);
    outptr += my_size;
    size += my_size;
  }
  gather(sizes, size);
  igatherv(outptr-size, sizes, hndl[i]);
  if(i>0) waitall(hndl[i-1], 1);
}
waitall(hndl[N/P/t], 1);
```

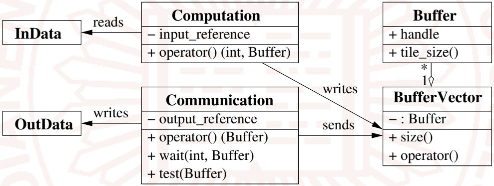
Parallel Compression - Adding a Window

```
for (i=0; i < N/P/t; i++) {
  my_size = 0;
  for (j=i; j < i+t; j++) {
    my_size += compress(i*t+j, outptr);
    outptr += my_size;
  }
  gather(sizes, my_size);
  igather(outbuf, sizes, hndl[i]);
  if (i > w) waitall(hndl[i-w], 1);
}
waitall(hnld[N/P/t-w], w);
```

Automatic Transformation

Templated Transformation

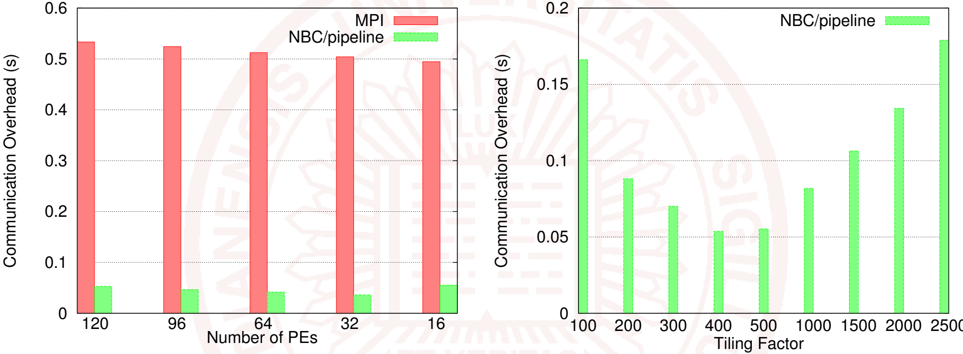
- requires buffer, computation and communication functor
- C++ template tiles loops and uses window
- $\Rightarrow$ programmer-directed overlap simplifies optimization



Two Examples

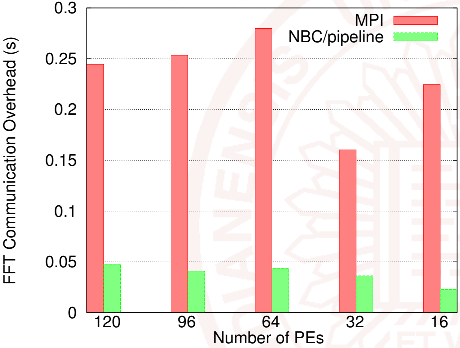
- parallel compression
- parallel 3d Fast Fourier Transformation

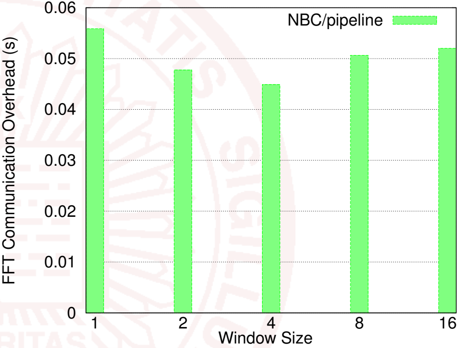
Parallel Compression



- 128 2 GHz Opteron 246 nodes, InfiniBand^{TM}
- 146MiB data compressed with `bzip2`
- 21% speedup on 120 PEs

3d Fast Fourier Transformation





- 16% speedup on 120 PEs
- weak scaling $(400^3, 480^3, \ldots, 720^3)$

Conclusions and Future Work

Conclusions

- loop-tiling and introduction of a commmunication-window to leverage non-blocking operations
- proposed a template-driven optimization scheme to assist the programmer
- showed the usefulness and performance advantages with two applications
- LibNBC and templates available at: `http://www.unixer.de/NBC`

Future Work

- optimize more (real-world) applications
- automatic parameter tuning

Conclusions and Future Work

Conclusions

- loop-tiling and introduction of a commmunication-window to leverage non-blocking operations
- proposed a template-driven optimization scheme to assist the programmer
- showed the usefulness and performance advantages with two applications
- LibNBC and templates available at: `http://www.unixer.de/NBC`

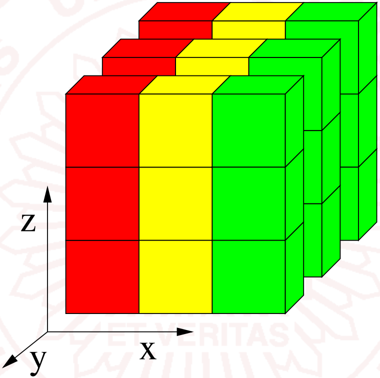
Future Work

- optimize more (real-world) applications
- automatic parameter tuning

Backup Slides

Transformation in z Direction

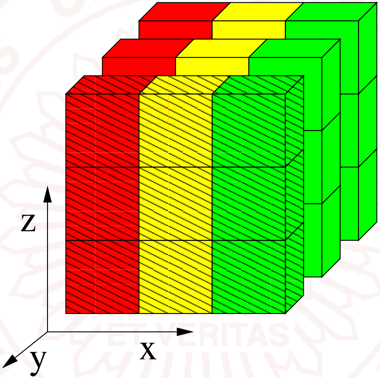
Data already transformed in y direction



1 block = 1 double value (3x3x3 grid)

Transformation in z Direction

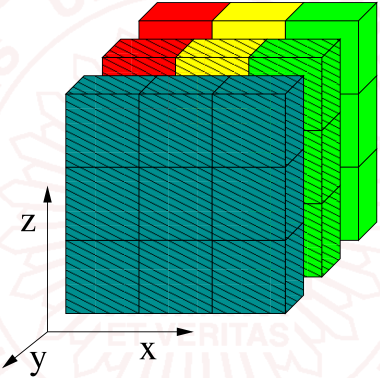
Transform first xz plane in z direction



pattern means that data was transformed in y and z direction

Transformation z Direction

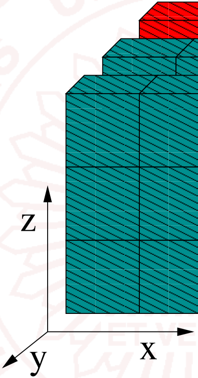
start MPI_Ialltoall of first xz plane and transform second plane



cyan color means that data is communicated in the background

Transformation in z Direction

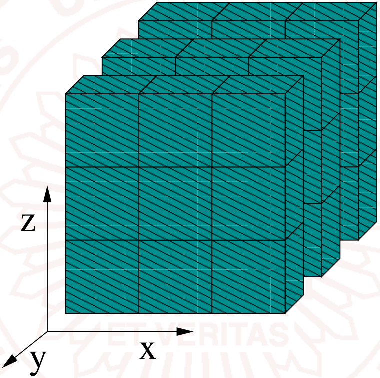
start MPI_Ialltoall of second xz plane and transform third plane



data of two planes is not accessible due to communication

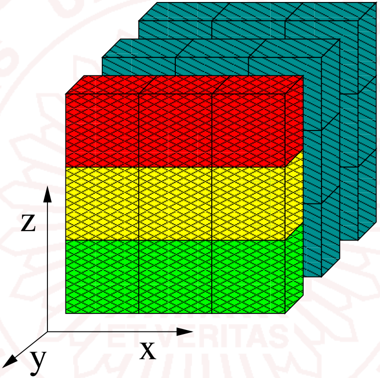
Transformation in x Direction

start communication of the third plane and ...



we need the first xz plane to go on ...

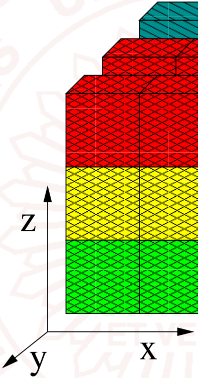
... so MPI_Wait for the first MPI_Ialltoall!



and transform first plane (new pattern means xyz transformed)

Transformation in x Direction

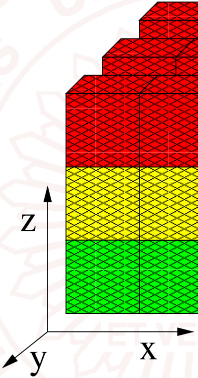
Wait and transform second xz plane



first plane's data could be accessed for next operation

Transformation in x Direction

wait and transform last xz plane



done! → 1 complete 1D-FFT overlaps a communication