Automatic MPI Application Transformation with ASPhALT

Anthony Danalis, Lori Pollock, Martin Swany
Overview

• Goal
  • High performance communication for MPI applications that is easy to achieve

• Solution
  • An automatic system that transforms simple communication code into more efficient code by improving the overlap of computation with communication

• Impact
  • Existing applications can enjoy improved performance
  • New applications can be written more simply and automatically optimized for various platforms
Overlapping Computation and Communication

(a) Non-blocking, CPU-based I/O
- Application
- OS Kernel
- Memory
- TCP/IP
- CPU

(b) Asynchronous, RDMA-based I/O
- Application
- I/O Library
- User space
- Kernel space
- CPU

Time Flow

Execution ➔ Data Transfer ➔ Overlapping ➔ Inter-Device Operation
Overlapping Details

• Minimize effective overhead of data movement by overlapping it with useful work
  • An old idea
  • Different approach than using large messages for high bandwidth

• What does it mean for parallel application development?
  • Post a send as soon as sufficient data is ready
  • Do useful work
  • Check status after completion (minimal polling or sleeping)

• Difficult to optimize, difficult to maintain
  • Particularly as platforms change
Overlapping Transformation - Simple Example

**Original code**

```fortran
integer, dimension(M,N):: array

do i = 1, N
    /* computation kernel */
    subroutine( array(1,i) )
enddo

size = M*N
DataTransferCall( array(1,1), size, ... )
Other_Computation()
```

**Tiled code**

```fortran
integer, dimension(M,N):: array

do i = 1, N, K
    do j = i, i+K-1
        /* computation kernel */
        subroutine( array(1,j) )
    enddo
    if( i > K ) then
        /* block for the arrival of the data */
        MPI_WAITALL( request( i - K ) )
    endif
    size = M*K
    /* asynchronous network transfer */
    MPI_ISEND( array(1,i), size, ... )
    MPI_ISEND( desנח( ... ), request(i), ... )
enddo

MPI_WAITALL( request( i - K ) )
```
ASPhALT
- Automatic System for Parallel Application Transformation
ASPhALT Framework

- Based on the Open64 compiler
  - Early work was based on Nestor and was Fortran77-only (Parco ‘05)
- Open64 uses intermediate representation known as WHIRL
  - WHIRL has 5 levels and the compiler works by progressively lowering from the highest to the lowest
- A WHIRL tree can be transformed and unparsed to high-level source code
  - At the highest two levels
Transformer Structure

asphalt_transformer

AST Manipulation

WHIRL AST

Unparser

Open64

WHIRL AST

Parser

Transformed Application

Original Application

C, C++, F95
Evaluation of Transformations

- Initial manual transformations to evaluate efficacy
  - A. Danalis, K. Kim, L. Pollock, M. Swany, "Transformations to Parallel Codes for Communication-Computation Overlap", SC05
- Two scientific applications as targets
  - Chem E. and Physics apps from UD
  - FFTW and MPI_ALLTOALL
- Created communicationless versions of the code
  - Normalized execution time
Evaluation of Automatic Transformation - Synthetic Kernel

Slowdown VS. Tile Size

interconnect: Ammasso, NP: 16, size: 1440x1440x48x16 Bytes
Evaluation of Automatic Transformation - Synthetic Kernel

interconnect: Myrinet-MX, NP: 48, size: 1440x1440x48x16 Bytes
Evaluation of Automatic Transformation - Application “visco”

interconnect: Myrinet-MX, NP: 48, size: 9216x2305x48x16 Bytes
Evaluation of Automatic Transformation - Application “visco”

Slowdown VS. Tile Size

interconnect: Myrinet-GM, NP: 24, size: 9216x2305x48x16 Bytes
System Benchmarks

Infiniband (hydra) -- NP=20 -- 5760x5760x5760

Normalized Time

- Application: Original Code
- Application: ASPhALT Optimized

Time (sec)

- Benchmark: Single MPI_ALLTOALL
- Benchmark: Overlapped Comm. Overhead

Size (bytes), K=size/2304

1e+03 1e+04 1e+05 1e+06 1e+07
System Benchmarks

Myrinet MX -- NP=48 -- 9216x2305x48

- Application: Original Code
- Application: ASPHALT Optimized

- Benchmark: Single MPI_ALLTOALL
- Benchmark: Overlapped Comm. Overhead

Normalized Time vs. Size (bytes)
Ongoing Efforts

- Apply technique to Scatter/Gather (C)
- Apply technique to large send “fission” (C)
  - Matching sends/recvs impossible without out of band information
- Use OpenFabrics APIs
  - DAPL
- Support for compilation of communication into lower-level routines
  - Abstract hardware details
  - Abstract protocol/library details
  - Abstract language issues (Fortran and pointers)
Acknowledgements

• UD Students
  • ASPhALT: Anthony Danalis, Aaron Brown, Andrew Gearheart, Magnus Johnsson
    • (alumni: Lewis Fishgold, Ki-Yong Kim)

• ASPhALT co-PI: Lori Pollock
Questions?