Algorithmic-Based Fault Tolerance for Matrix Multiplication on Amazon EC2

Grey Ballard, Erin Carson, Nick Knight
CS 262a Fall 2009

Summary
- Our main goal is to study the feasibility of using the cloud for scientific computing
- Are the availability and elasticity of the cloud worth the sacrifice in performance?
- Due to the high performance variability of cloud computing, algorithms need to be flexible and resilient
- Large scale computations will likely encounter faults/stragglers
- We explore matrix multiplication as a case study in fault tolerance

Experiments on the Cloud

Amazon EC2
- Instances run as virtual machines
- We used the “small” instance type: each has 1 virtual core and 1 EC2 Compute Unit, “the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or Xeon processor”
- We built a virtual Linux cluster and configured it to support the Message Passing Interface (MPI)

Sequential Matrix Multiplication
- Measured sequential performance by benchmarking sequential matrix multiplication (DGEMM)
- Compared performance of implementations tuned for Intel platforms (Intel’s Math Kernel Library) with an autotuned implementation (ATLAS)

Communication Costs
- We measured the time required to send messages via MPI between pairs of instances in our cluster

Parallel Matrix Multiplication
- Measured parallel performance by benchmarking two MPI-based matrix multiplication implementations
- Compared performance of ScaLAPACK’s PDSYMM with the pipelined Scalable Universal Matrix Multiply Algorithm (SUMMA) on cluster of 25 instances
- Used MKL for sequential multiplication

Performance Overhead
- Given \( p^2 \) processors, \( 2p \) – 1 are dedicated to checksum data
- This overhead decreases as \( p \) increases
- Recovery consists of
  - Detecting a fault and respawning the process
  - Recovering the lost \( A, B \) data
  - Allowing the pipeline to empty (to reach a consistent state)
  - Recovering the lost \( C \) data
  - Refilling the pipeline

Our Implementation
- We implemented a C/MPI version of SUMMA and modified it to handle simulated faults
- We ran a “controller” process that listened for notifications while waiting on (non-blocking) MPI sends and receives
- Our first implementation, based on fault detection via timeouts, had trouble discerning faults from communication delays

Algorithmic-Based Fault Tolerance
- We seek parallel algorithms which persist throughout a computation
  - ABFT enables computation on untrustworthy nodes
  - Data lost during faults is recovered on-the-fly
  - Low overhead in the absence of faults

ABFT SUMMA
- SUMMA computes \( A \cdot B = C \) as a series of rank-k updates
- Communication is organized into pipelined ring broadcasts of panels of data along processor rows and columns
- We achieve fault tolerance by adding an extra (block) row or column to each input matrix and computing a checksum, and allowing SUMMA to compute

\[
\begin{bmatrix}
A \\
C_i(A)
\end{bmatrix} \cdot
\begin{bmatrix}
B \\
C_i(B)
\end{bmatrix} =
\begin{bmatrix}
C \\
C_i(C)
\end{bmatrix} \cdot
\begin{bmatrix}
C_i(C) \\
C_i(C_i(C))
\end{bmatrix}
\]

Weak Scaling of FT-SUMMA
- Standard SUMMA is weakly scalable
- Due to the low overhead, our implementation scales well (and approaches the performance of standard SUMMA)
- This experiment was run with local block size held constant at 500

Future Work
- Incorporate FT-MPI into our implementation
- Test performance on other (more expensive) instance types
- Apply ABFT to other matrix computation routines