What are our Options for Optimizing Numerical Weather Prediction Codes on Modern Supercomputers?

Dr. Daniel B. Weber
Center for Analysis and Prediction of Storms
University of Oklahoma

Acknowledgements

- Dr. Henry Neeman, Tiling development
- Scott Hill, Software Support (Papi, etc...)
- Dr. John Romo, NEC

Before we get started...

- I am a meteorologist, not a computer scientist!!! Computers help us perform science!
- Scientists need assistance with the terminology and concepts
- The current analysis is part of an on-going study of NWP codes (ARPS,WRF, research codes) running on commodity based clusters
- Applicable to other disciplines!!!

Outline

- Why optimize?
- Motivation and example application
- Optimization game plan
- Results
- Tiling
- Future work

Why Optimize?

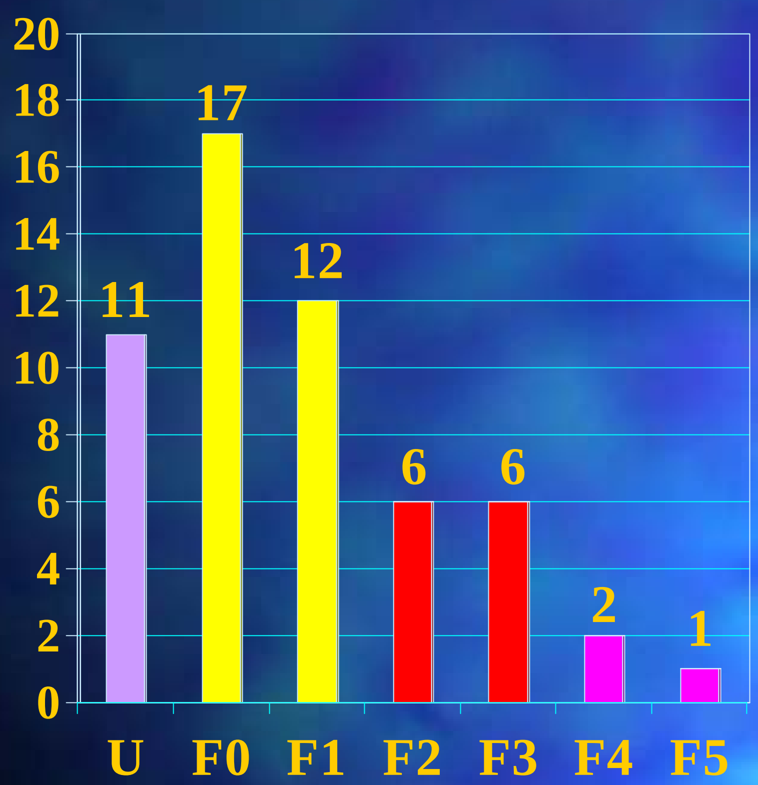
- Numerical Weather Prediction codes are very inefficient on the current HPC platforms
- High-resolution forecasts on a Continental US domain (<1km grid spacing)
- Generate ensembles of storm-scale predictions - provide probabilities of specific weather events for public use
- Improve warning lead times
- The usefulness of numerical weather prediction depends on the efficient use of large clusters

Motivation

- Adverse weather impacts the US economy and the lives of you and me... (on the order of billions annually)
- We can reduce losses of both life and property
- Can numerical weather prediction improve storm forecast and warning quality?

May 3 Tornado Damage

Largest Tornado Outbreak In Oklahoma History



- 60+ tornadoes statewide
- Previous state record 26 tornadoes in one day
- First F5 since 1982
- First F5 for Oklahoma City

Courtesy Dave Andra, Oklahoma City Area National Weather Service Forecast Office

The Forecasts and Warnings

- Severe thunderstorms in 4:30 AM forecast
- Thunderstorm outlooks mention tornadoes
- First warnings issued SW of Oklahoma City 4:15 PM
- Short term forecast at 5:40 PM mentions tornadic storms moving into metro by 7:00 PM
- Numerous warnings and radar observations/detailed statements tracked tornado into and through metro area
- NOTE: models did not forecast the event!

Courtesy Dave Andra, Oklahoma City Area National Weather Service Forecast Office

Application: The ARPS Model

- Grid point model solving the Navier-Stokes Equations using Fortran 90, MPI and 150 3-D arrays
- Large time step solver (temperature, watar quantities, turbulence, gravity wave, advection)
- Small time step solver (velocities, pressure, sound waves)
- Many small time steps per large time step

Sample Equation Set

$$\frac{\partial u}{\partial t} = -(u\frac{\partial u}{\partial x} + v\frac{\partial u}{\partial y} + w\frac{\partial u}{\partial z}) - \frac{1}{\rho}\frac{\partial p}{\partial x} + Turb + Cmix$$

$$\frac{\partial v}{\partial t} = -(u\frac{\partial v}{\partial x} + v\frac{\partial v}{\partial y} + w\frac{\partial v}{\partial z}) - \frac{1}{\rho}\frac{\partial p}{\partial y} + Turb + Cmix$$

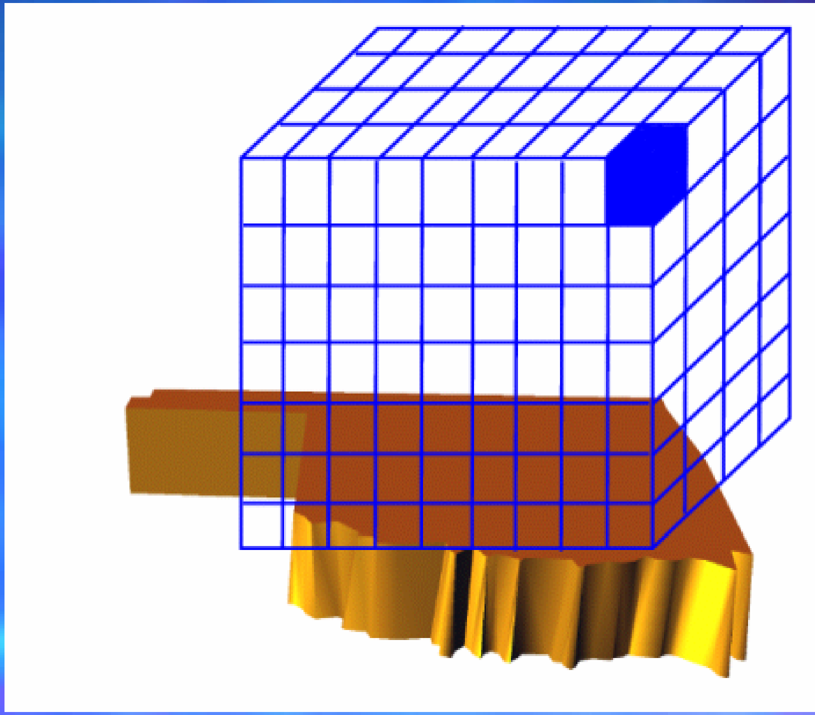
$$\frac{\partial w}{\partial t} = -(u\frac{\partial w}{\partial x} + v\frac{\partial w}{\partial y} + w\frac{\partial w}{\partial z}) - \frac{1}{\rho}\frac{\partial p}{\partial z} + Turb + Cmix$$

$$\frac{\partial p}{\partial t} = -(u\frac{\partial p}{\partial x} + v\frac{\partial p}{\partial y} + w\frac{\partial p}{\partial z}) - \rho(\frac{\partial u}{\partial x} + \frac{\partial v}{\partial y} + \frac{\partial w}{\partial z})$$

- Plus 8 more equations!

Numerical Weather Prediction Approach

- Break the forecast into grid boxes (finite grid)
- Solve complicated equations within each grid box to account for
  - wind speed and direction
  - pressure, temperature
  - radiative processes
  - surface vegetation
  - lakes and oceans
  - clouds, rain, hail, snow
  - turbulence



Parallel Implementation

- Spread domain over many processors to reduce the time required to run the forecast (smaller domains = fewer calculations = less time computing)
- Use Message Passing Interface (MPI), remember this is a real time application!
- MPI sends messages to update the solution along the interior processor boundaries

Numerical Weather Prediction

- Over the course of a single forecast, the computer model performs billions/trillions of calculations
- Requires the fastest supercomputers in the world -- capable of performing trillions of calculations each second
- Local cluster (OSCER), photo courtesy of H. Neeman

Continental US Thunderstorm Forecasts

- Computer resource estimate (1-km grid spacing)
  - 5500x3600x100grid points x 3500 calc/grid point = 6.9 TFLOPS
  - Current cluster technology (i.e. IA-32 based) 3GHz Pentium 4 provides a peak of 6 GFLOPS/processor
  - Requires 1155 processors assuming a perfect CPU utilization, network, and message passing environment

Review of
Computer Processing Types:

Scalar vs. Vector

Vector Processor Architecture

- Single instruction/multiple data
- Fast access to memory
- Streamlined computation units specialized for floating point arithmetic
- Poor integer performance, common scalar architecture chips are used
- Very expensive due to fast memory and highly specialized processors
- Vector codes are 80% efficient

Scalar Processor Architecture

- Single instruction/single data
- Variants include superscalar (multiple functional units)
- Inexpensive and fast sequential processors (Clock rates > 3GHz)
- Slower memory access than vector architecture
- Scalar processors utilize multi-layer cache to minimize memory access latency
- Scalar codes are only 10-20% efficient

Supercomputer Evolution

- In the past, NWP centers used CRAY C90's (NCEP) and Fujitsu VPP 700 series (ECMWF) vector supercomputers
- NCEP and ECMWF upgraded to scalar architecture based clusters (IBM P690's)
  - Significant tuning requires many man hours (months)
- RESULT: Weather applications REQUIRE modified code to run efficiently on scalar technology due to a slow memory sub-system and less productive functional units

Must Adopt a Scalar Optimization Strategy

- Assess the code, isolate the subroutines requiring the most cycles (apply performance tools e.g. Perfex, Speedshop, PAPI, Apprentice)
- Is the code memory bound or compute bound
- Memory access is slow, need to maximize data reuse
- Rethink the order/layout of the computational framework?? (time consuming and buggy)
- The yardstick: scalar processor efficiency will be compared to vector processor efficiency!

Optimization Game Plan

Optimization Guidelines

- Keep the code easy to read, important for code maintenance and further development (Meteorologists developed the code)
- The modified code must perform well on both vector and scalar architectures (keep do loops vectorizable)
- Can we achieve 80% efficiency on a single scalar-type processor?

Potential Areas for Optimization

- Message passing
- File i/o
- The number of calculations (many ways...include tuned libraries, pre-compute intermediate terms, merge loops)***
- Calculation overhead (memory references, instructions)***
- Compiler optimizations - don't forget this valuable option!!!

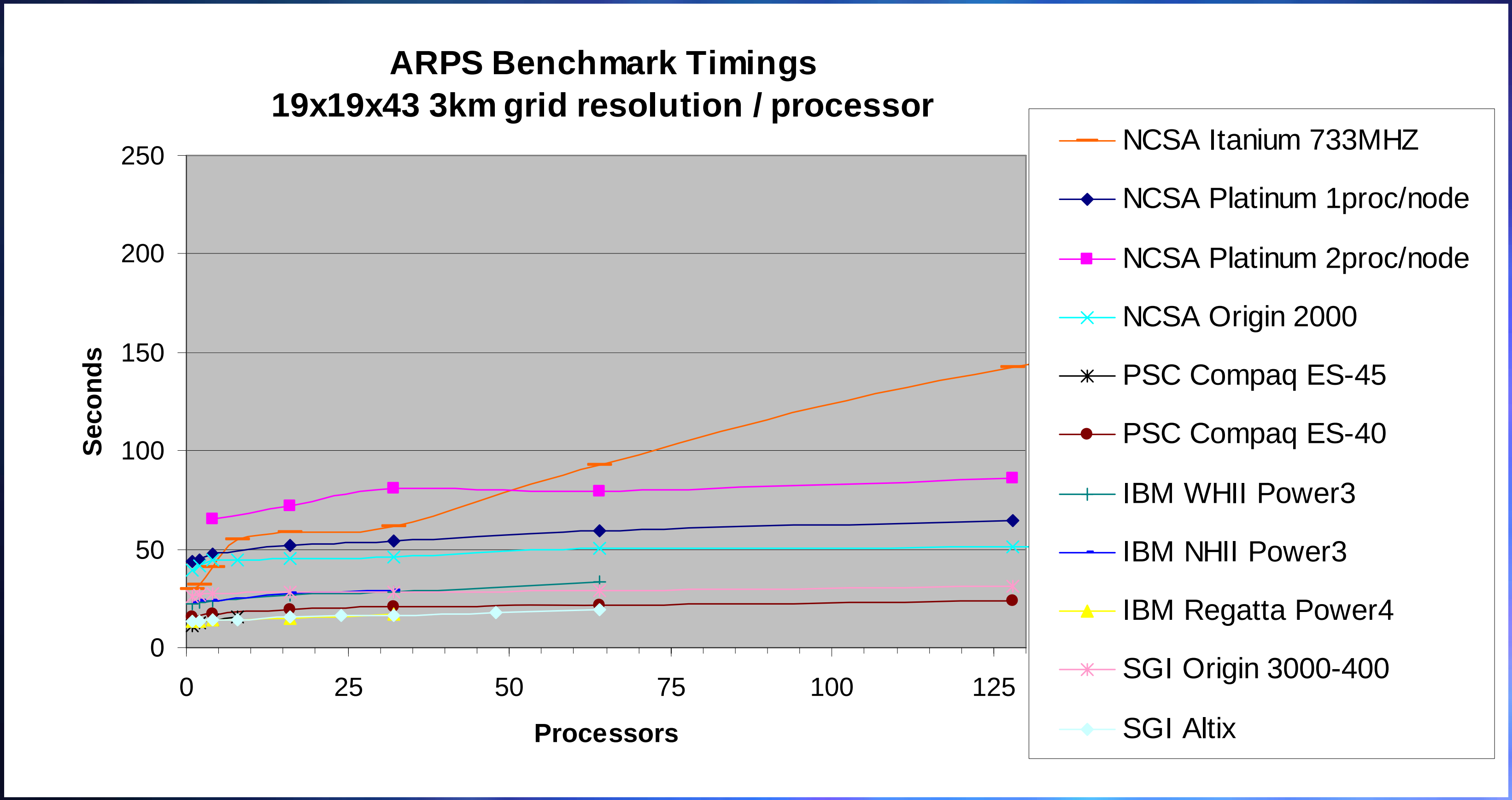
MPI Optimization

- Reduce the number messages sent and received (latency and bandwidth)
- Reduce the size of the messages (bandwidth)
- If you have to choose between #1 or #2 above, select #1 (if not bandwidth limited)
- Hide communications (limited to the number of calculations (latency/bandwidth)
- Redesign your code to reduce the number of messages, group messages together (include another fake zone if needed) (lat/band)

MPI Optimization

- ARPS has > 50 mpi sends in a single big time step
- Could be reduced to 3 sends if the number of fake zones was increased by one in each horizontal direction (research code)
  - Removes intermediate sends for advection, mixing, computational mixing
  - Combines sends
  - Simplifies the code
  - Cost = additional computations

ARPS Benchmarks



Early Results from Optimization Efforts

ARPS Optimization History

- 15 years of code development and optimization
- A focused effort during 1997-1998 yielded 20-33% improvement on computers ranging from IA-32 to Vector processors (combined loops and saved redundant calculations into arrays, etc...)
- Optimization of the ARPS on the SX-5 platform applied loop fusion (inner do loop limits = array limits - manually change do loops!!!!!)
- Loop fusion increases performance, in terms of FLOPS, up to 600% on a vector machine for small vector length problem sizes (inner loop < processor vector length) prevents chain breakage

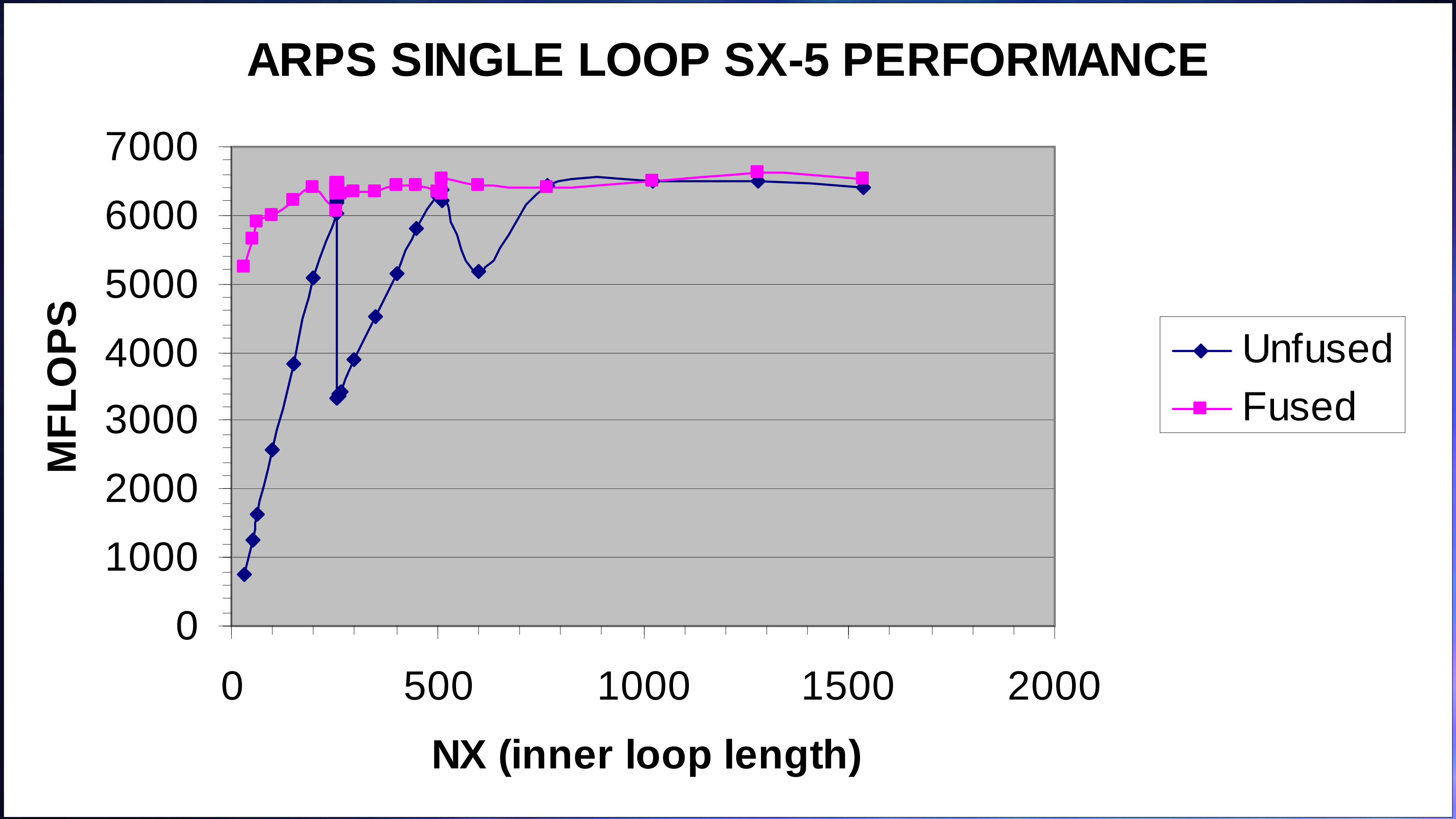
Vector Application: Fused Loop

- Extend the do loop limits to the full array size

```
do k=1,nz-1
   do j = 2,ny-1    ! Old limits
      do i = 2,nx-1
      a(i,j,k) = (b(i+1,j,k)-b(i,j,k))*0.5*dxinv
end do's
do k=1,nz-1         ! Note outer loop is untouched!
   do j = 1,ny       ! New limits
      do i = 1,nx
       a(i,j,k) = (b(i+1,j,k)-b(i,j,k))*0.5*dxinv
end do's
```

ARPS SX-5 Optimization Results

SX-5 Peak is 8000 MFLOPS



Loop Merging

- Combine loops, the result is reduced loads and stores. This is very important on scalar processor technology
- Need to understand the order of execution of the code, this requires a detailed knowledge of the physical processes etc
- Code restructuring is a man-power intensive process
- Compute forcing: $$uforce = uforce + u \frac{\partial u}{\partial x}$$

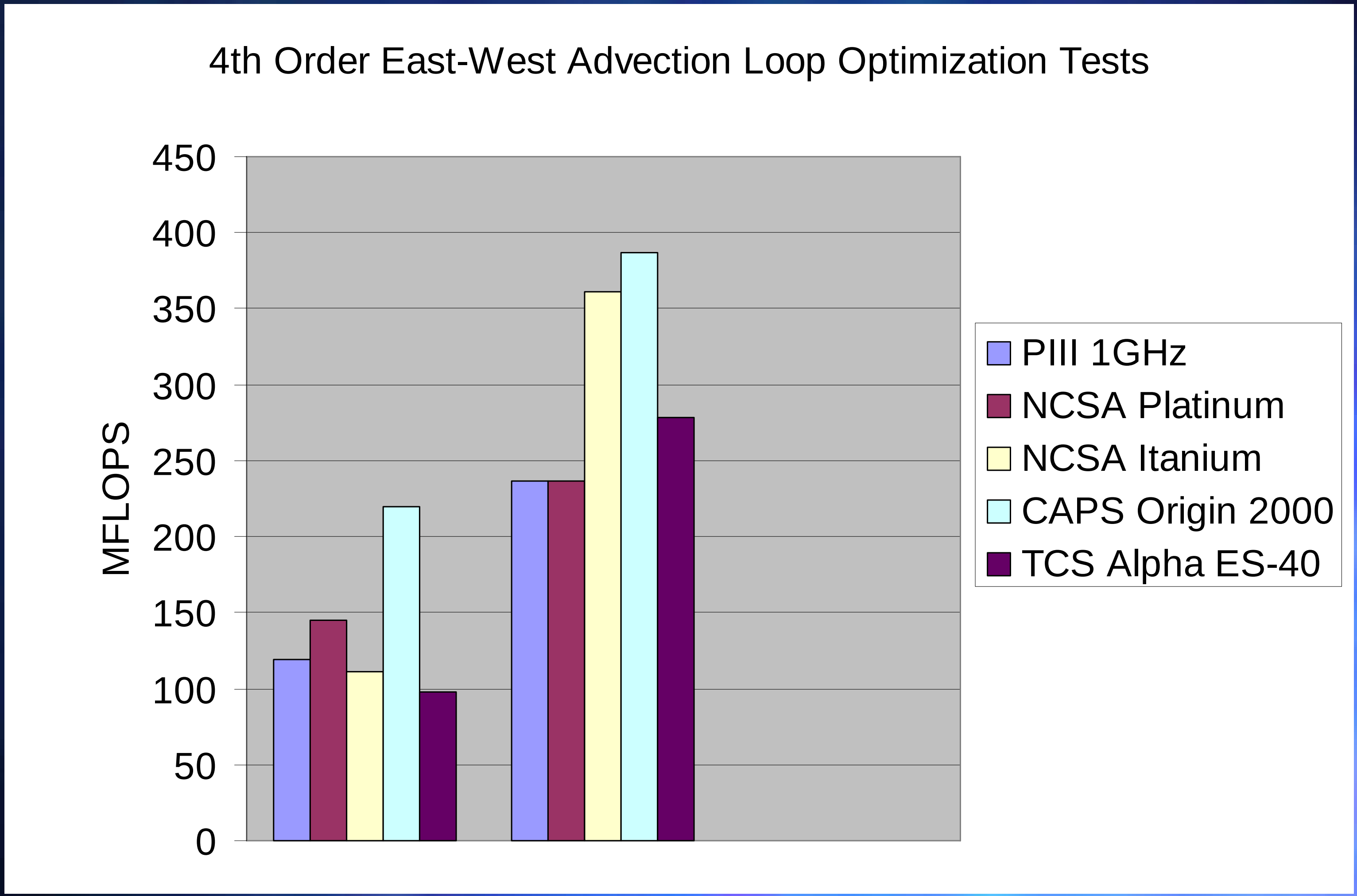
Example: Horizontal 4th Order Advection

```
DO k=2,nz-2          ! compute avgx(u) * difx(u)
DO j=1,ny-1              !  Total of 14 flops...
DO i=1,nx-1
   tem2(i,j,k)=tema*(u(i,j,k,2)+u(i+1,j,k,2))*(u(i+1,j,k,2)-u(i,j,k,2))
END DO's
DO k=2,nz-2          ! compute avg2x(u)*dif2x(u)
DO j=1,ny-1
DO i=2,nx-1
   tem3(i,j,k)=tema*(u(i-1,j,k,2)+u(i+1,j,k,2))*(u(i+1,j,k,2)-u(i-1,j,k,2))
END DO's
DO k=2,nz-2           ! compute 4/3*avgx(tem2)+1/3*avg2x(tem3)
DO j=1,ny-1           ! signs are reversed for force array.
DO i=3,nx-2
      uforce(i,j,k)=uforce(i,j,k)
:                +tema*(tem3(i+2,j,k)+tem3(i-1,j,k))
:                -temb*(tem2(i-1,j,k)+tem2(i,j,k))
END DO's
```

- Horizontal Advection - Modified Version
- Three loops are merged into one large loop that reuses data and reduces loads and stores

```
 DO k=2,nz-2              ! Total of 18 flops, 4 additional flops...
 DO j=1,ny-1
 DO i=3,nx-2
     uforce(i,j,k)=uforce(i,j,k)
:        +tema*((u(i,j,k,2)+u(i+2,j,k,2))*(u(i+2,j,k,2)-u(i,j,k,2))
:        +(u(i-2,j,k,2)+u(i,j,k,2))*(u(i,j,k,2)-u(i-2,j,k,2)))
:       -temb*((u(i,j,k,2)+u(i+1,j,k,2))*(u(i+1,j,k,2)-u(i,j,k,2))
:        + (u(i-1,j,k,2)+u(i,j,k,2))*(u(i,j,k,2)-u(i-1,j,k,2)))
 END DO's...
```

Result - Merged Loops



Further Optimization

The Case for Tiling

Tiling

- Tiling can be defined as the process to which the original domain of computation is split up into smaller sections that can fit into the top level cache
- The goal of tiling is to reuse data in the L2 (or L3) cache as much as possible prior to computing the next region
- This approach requires the changing of do loop limits to perform calculations on the sub-domain
- The goal is to tune the application to fit the tile region within the cache and achieve enhanced data reuse and application performance
- Tiling will work if your code is MEMORY BOUND

Memory Requirement Analysis

- Assess the use/reuse of arrays in your code
- Tile regions of significant reuse of data
- Determine the optimal tile size for each tiled region of code (different parts of the code contain a different number of arrays used)
- Automate the tiling determination (some regions require smaller tiles (more arrays) than other areas

Tiling example

```
DO bigtimestep = 1,numbigsteps
  do tile
    solve potential temperature, water quantities (clouds, water
    vapor, precipitation) turbulence, gravity waves, advection
  end tile                                        40 3-d arrays...
  call mpiupdate....update pt, turb, water
    Do smallstep= 1,numsmallsteps
          do tile
            solve u,v (horizontal wind field)     5 3-d arrays...
          end tile
          call mpiupdate...update u and v
          do tile
            solve w (vertical velocity) and pressure  20 3-d arrays
          end tile
          call mpiupdate...update w and p
    End smallstep
 End bigtimestep
```

Tiling Test Description

- GOAL: Develop a strategy for modifying the forecast model to achieve better scalar architecture single processor performance
- The test loop is similar to fourth order computations (advection, computational mixing) and turbulent mixing
- Use PAPI to access the performance counters on my Dell Pentium III laptop
- Evaluate L1 and L2 cache instruction and data misses and loads/stores as well as FLOPS and the Translation look aside buffer (TLB) as a function of problem size
- Adjust problem size to assess memory hierarchy behavior and patterns

Tiling Code Description

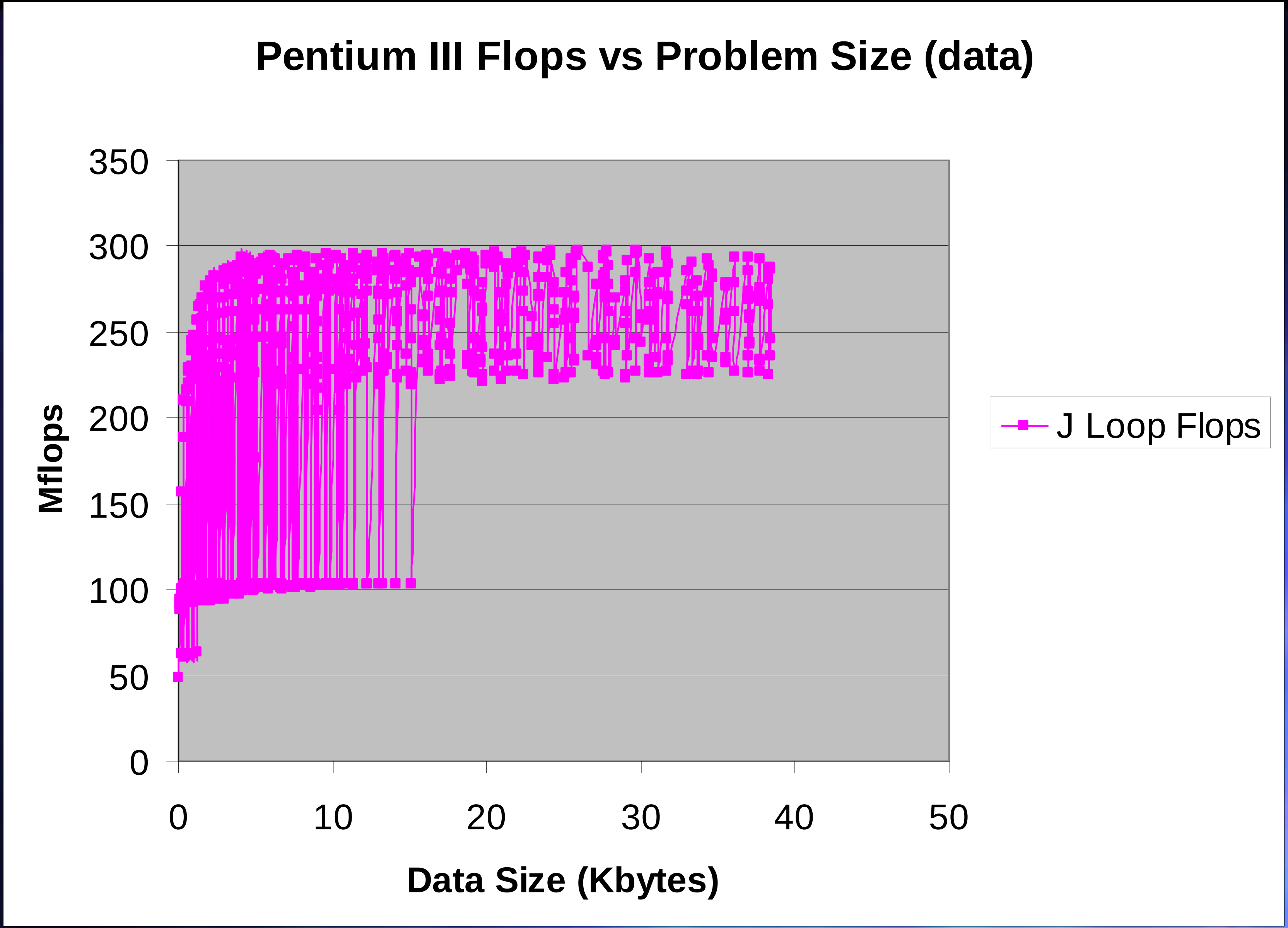
- I Loop (Fortran 90)
  - Outer loops increment nz,ny,nx... from small to large

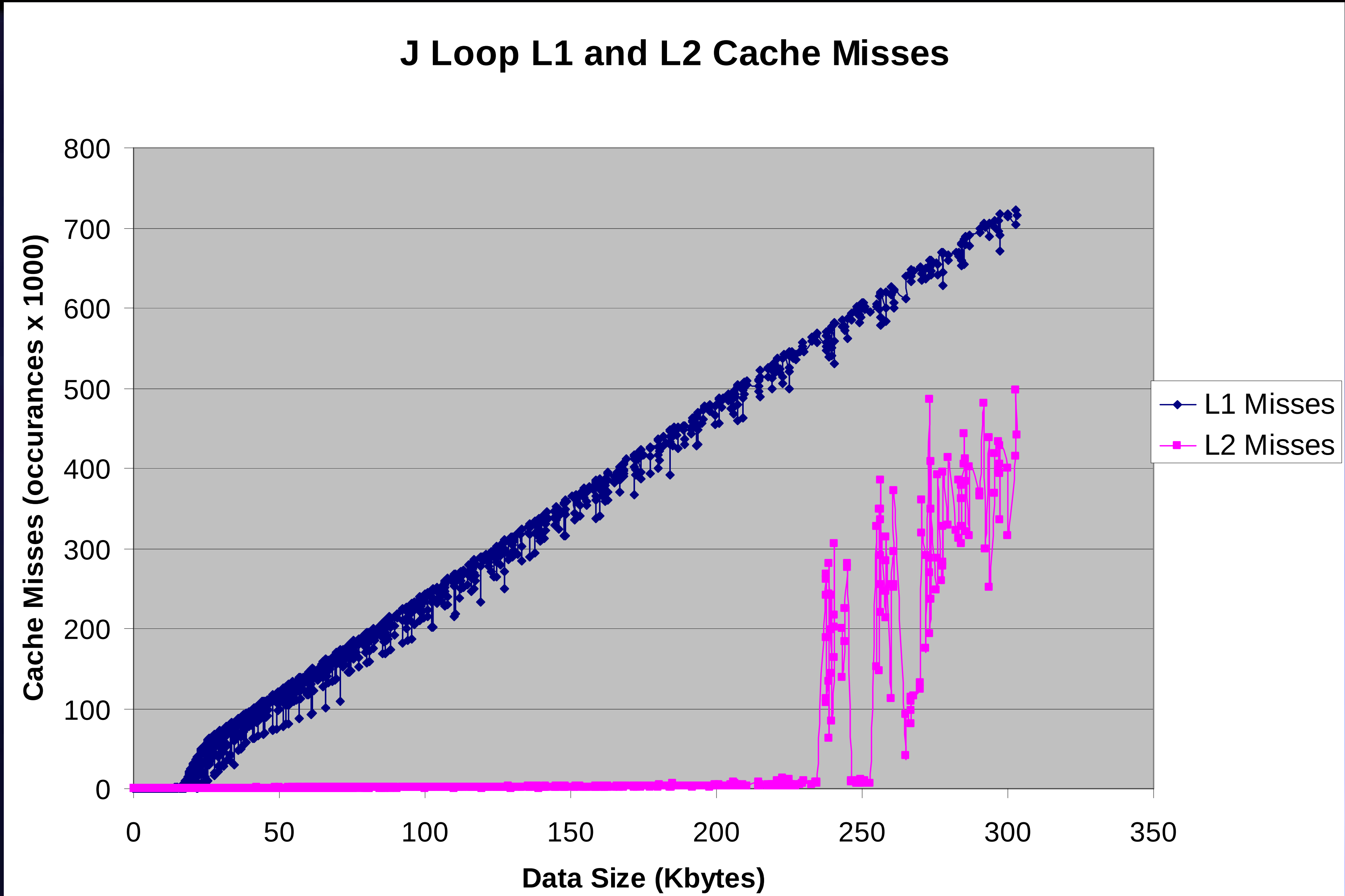
```
Do n = 1,loopnum      !  Loopnum = 80
Do k = 1,nz
Do j = 1,ny
Do i = 3,nx-2
Pt(i,j,k) = (u(i+2,j,k)+u(i+1,j,k)-u(i,j,k)+u(i-1,j,k)-u(i-2,j,k))*1.3*n
End Do's
```

- 5 point stencil, reusing data in the i-direction
- Nx,ny,nz are varied to adjust the size of the problem
- U and PT are allocated and deallocated for each change in nx, ny, and nz
- Same tests were performed for reusing data in j and k directions

Research Code Optimization

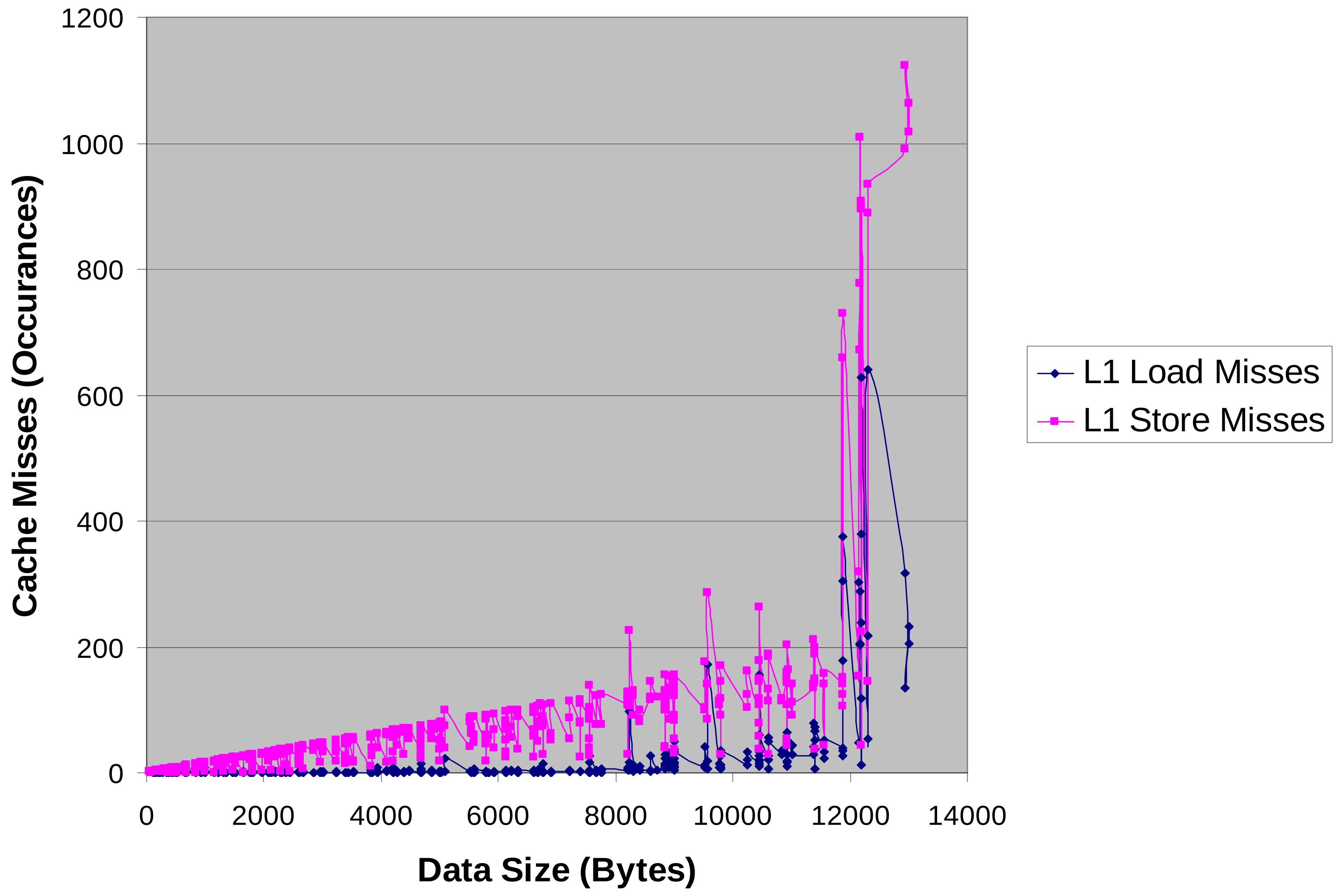
Tiling Results


Pentium III Flops vs Problem Size (data)
Mflops
350
300
250
200
150
100
50
0
0
10
20
30
40
50
Data Size (Kbytes)
J Loop Flops



J Loop L1 and L2 Cache Misses
Cache Misses (occurances x 1000)
800
700
600
500
400
300
200
100
0
0
50
100
150
200
250
300
350
Data Size (Kbytes)
L1 Misses
L2 Misses

J Loop L1 Cache Load and Store Misses



Tiling Experiments Summary

- Performance (FLOP rating) of scalar architecture is linked to the length of the inner most loop, larger inner loop ranges utilized data in the L1 cache more efficiently – similar to VECTOR architecture behavior!
- Loop performance >40% of peak for problem data sizes < L2 cache
- FLOP rating not dependant of the data size with respect to the L1 cache
- Significant reduction of throughput (factor of 2) was observed when the data size > L2 cache

Future Work

- Implement tiling in the small time step (completed – 20% improvement in the small time step efficiency)
- Implement tiling on the large time step (potential big win on large time step calculations)
- Investigate the behavior of loop length (can loop fusion assist the compiler in optimizing the loop?)
- Modify the code to include do loop limits as parameters set at run time
- Investigate the possibility of restructuring the order of computation (to reduce memory references)

Thank you for your attention!

CAPS Forecast System
visit: www.caps.ou.edu/wx

Email: dweber@ou.edu