New Advances in the Gyrokinetic Toroidal Code and Their Impact on Performance on the Cray XT Series

Nathan Wichmann, Cray Inc.

Mark Adams, Columbia University

Stephane Ethier, Princeton Plasma Physics Laboratory

Outline

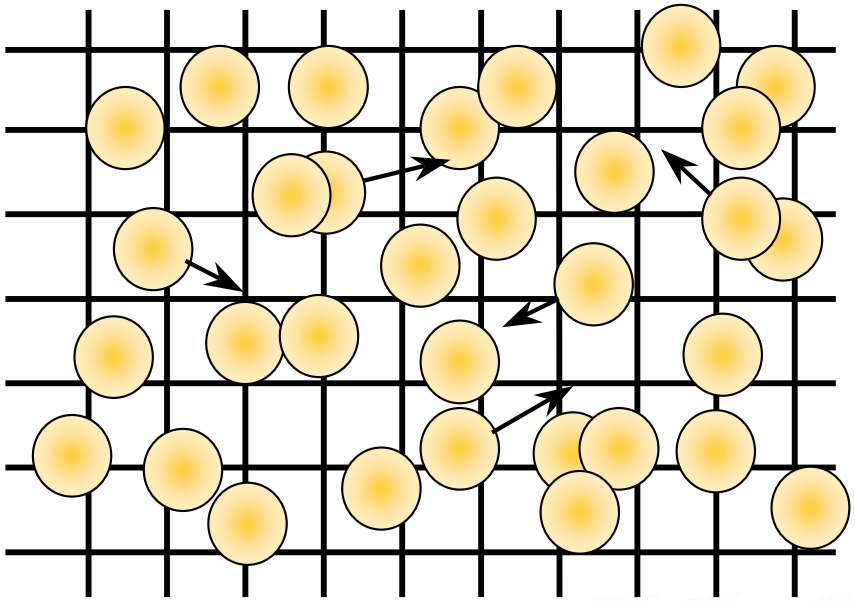
- Intro to GTC
- Major Components
- New Decomposition Strategy
- Goals
- Results
- Future Work
- Conclusions

The Gyrokinetic Toroidal Code GTC

- **Description:**
  - 3D Particle-in-cell code (PIC) in toroidal geometry
  - Developed by Prof. Zhihong Lin (now at UC Irvine)
  - Used for non-linear gyrokinetic simulations of plasma microturbulence [Lee, 1983]
  - Fully self-consistent
  - Uses magnetic field line following coordinates (ψ,θ,ζ) [Boozer, 1981]
  - Grid follows the magnetic field lines (twisting around the torus)
  - Guiding center Hamiltonian [White and Chance, 1984]
  - Non-spectral Poisson solver [Lin and Lee, 1995]
  - Low numerical noise algorithm (δf method)
  - Full torus (global) simulation

Particle-in-cell (PIC) method

- Particles sample distribution function (markers).
- The particles interact via a grid, on which the potential is calculated from deposited charges.

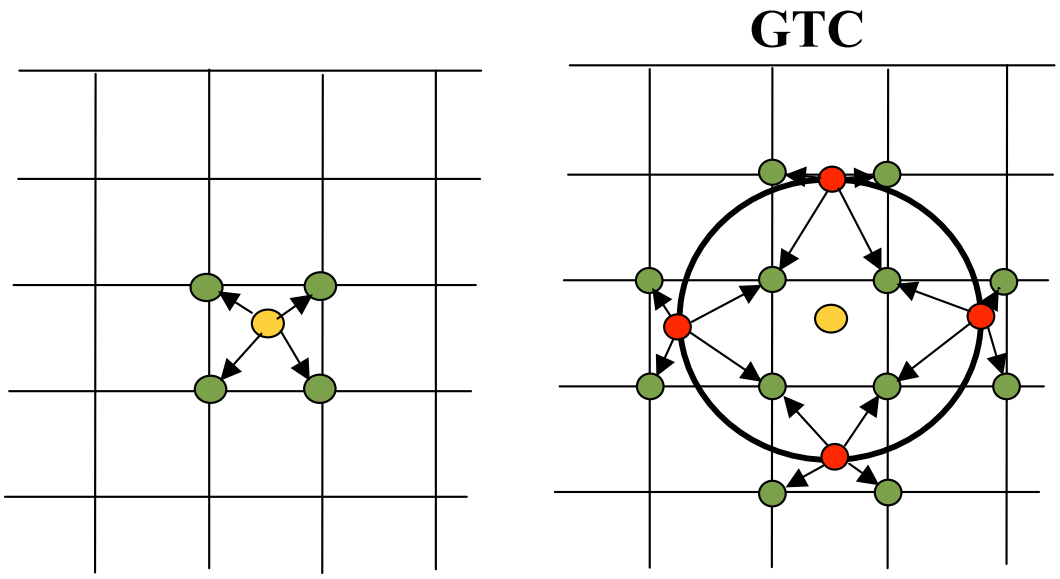


The PIC Steps

- "SCATTER", or deposit, charges on the grid (nearest neighbors)
- Solve Poisson equation
- "GATHER" forces on each particle from potential
- Move particles (PUSH)
- Repeat…

Charge Deposition for charged rings: 4-point average method

Charge Deposition Step (SCATTER operation)

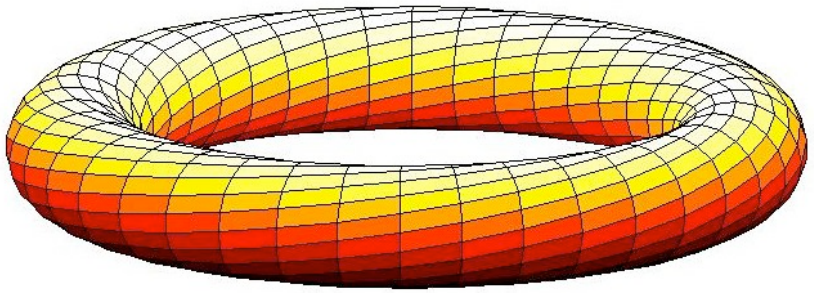


Classic PIC

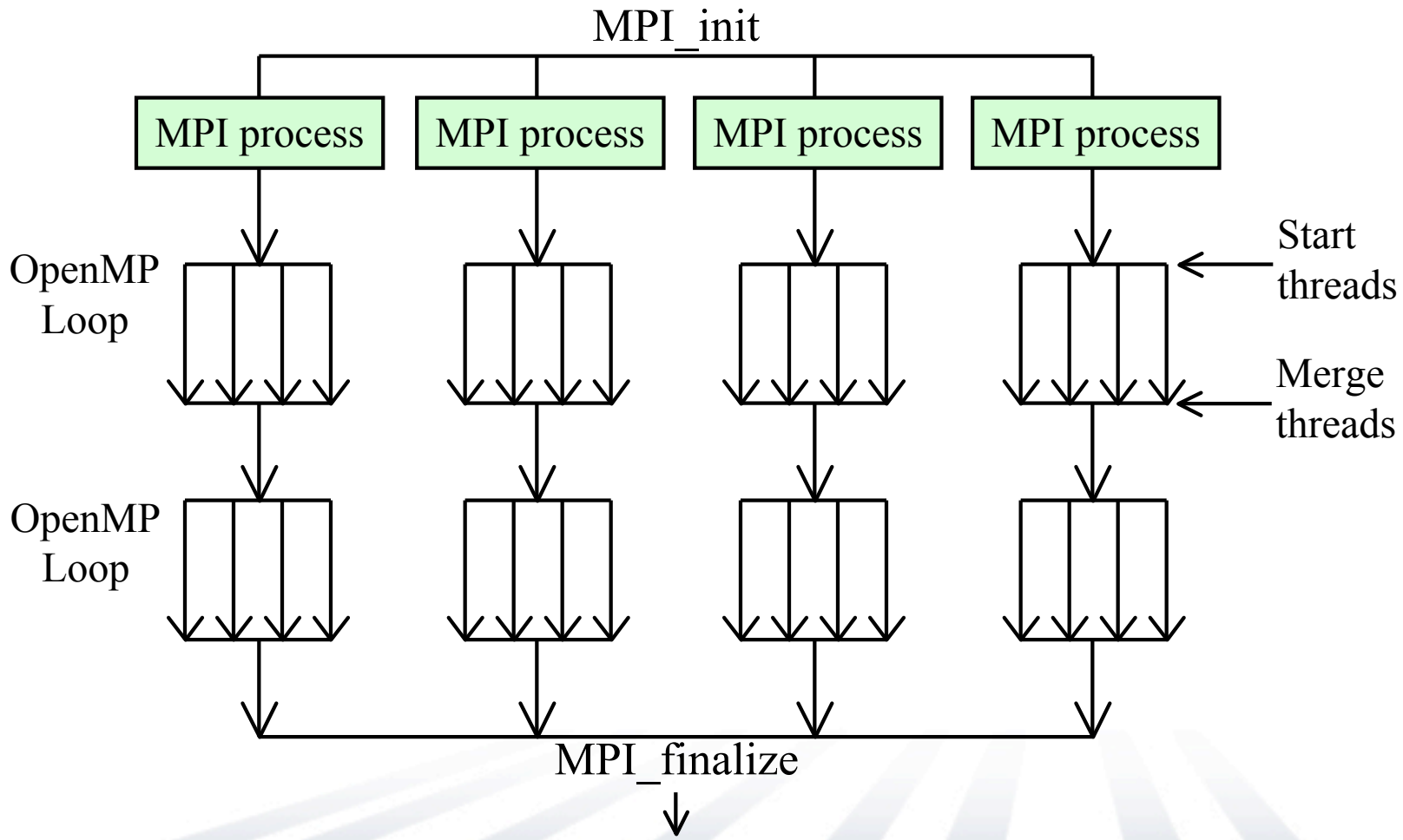
4-Point Average GK (W.W. Lee)

Domain Decomposition: Pre 2007

- Domain decomposition:
  - each MPI process holds a toroidal section
  - each particle is assigned to a processor according to its position
- Initial memory allocation is done locally on each processor to maximize efficiency
- Communication between domains is done with MPI calls (runs on most parallel computers)



2nd Level of Parallelism: Loop-level with OpenMP



Computational Facts about GTC

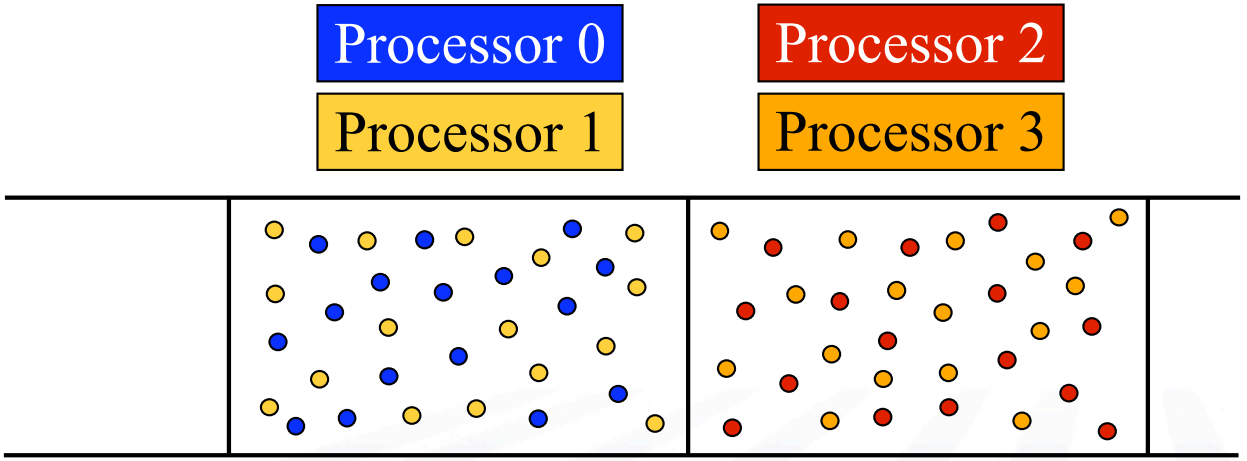
- Only 5000 lines.
- Written in Fortran 95
  - Latest version uses object oriented programming
- Highly portable. GTC runs on most parallel computers as long as the MPI library is available.
- Part of the NERSC benchmark suite of codes to evaluate new computers.
- Pre 2007 version ran using real(4) (32 bit reals)
- New version using real(8)
  - PETSc solver needs 64 bit precision
  - Rest of gtc only needs 32 bit precision but currently uses 64 bit

Major component of GTC

- Pushi
  - Major computational kernel "Moves particles"
  - Large body loops; Gathers; Lots of loop level parallism
- Chargei
  - Major computational kernel "Scatter"
  - Gather/Scatter with potential conflicts; Can be restructured to exploit loop level parallelism
- Shifti
  - Sorts out particles that move out of its domain and sends those to the "next" processor. Process repeats until all particles are where they are suppose to be.
- Poisson Solver
  - Solve Poisson Equation. Prior to 2007 the solve was redundantly executed on each processor. New version uses the PETSc solver to efficiently distribute the work.
- Smooth and Field
  - Smaller computational kernels

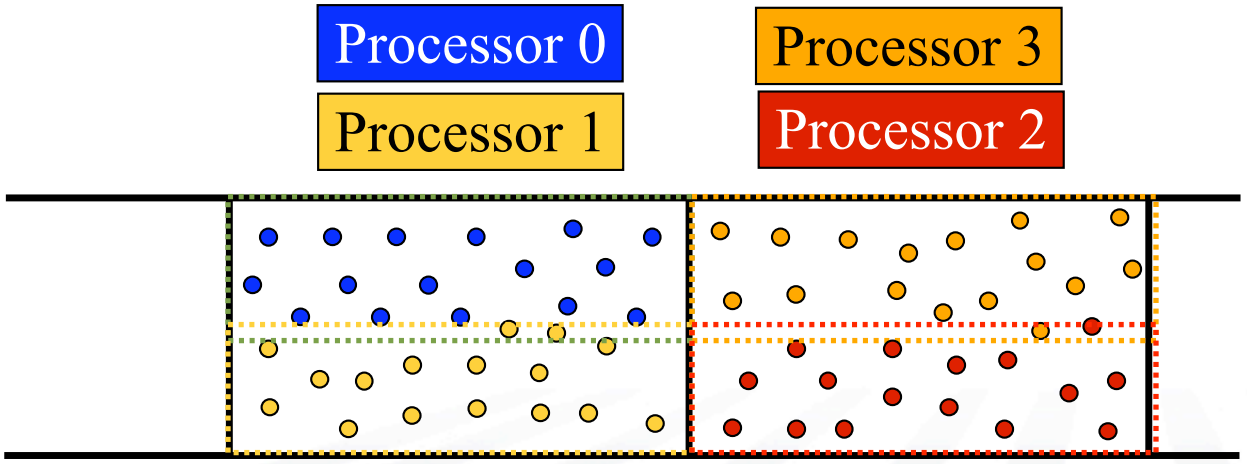
GTC pre-2007: 1D Particle decomposition

- Each domain in the 1D domain decomposition can have more than 1 processor associated with it.
- Each processor holds a fraction of the total number of particles in that domain.
  - Required an All_reduce to collect all of the charges from all particles in a given domain
- Each processor stores the mesh of an entire plane
  - Poor cache characteristics
  - Redundant work: “smoothing”, “field”, and “poisson” calculations not parallel
  - Not scalable to large reactor sizes.



GTC 2007 and beyond: 2D Particle decomp

- Each processor holds a fraction of particles - and of grid
  - Radial geometric partitioning for equal area per processor
  - Memory footprint scalable to LARGE reactor sizes
- Significant difference in communication
  - Large All_reduce has been eliminated
  - Require extra shift to move particles in the radial direction
- Domain overlap due to
  - Discrete nature of grid (ie, not aligned with radial partitions)
  - Gyroradius lead to deposition on larger grid



Goal: Develop a new baseline benchmark

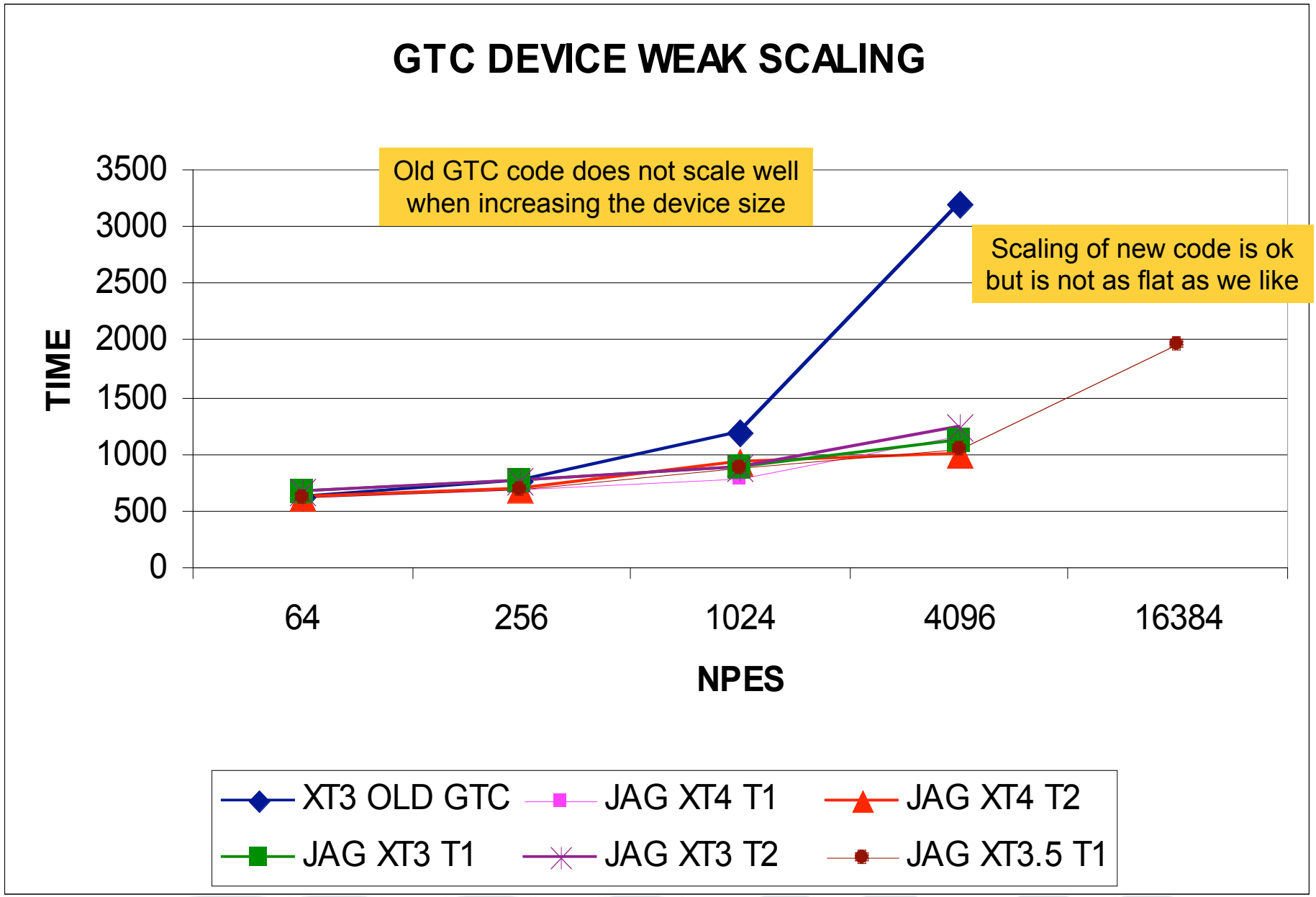
- GTC has change significantly in both capabilities and performance characteristics
  - Can now simulate much larger reactors
  - Now uses PETSc for the solver
  - Major all_reduce has been eliminated but a “shift” was added
- Want a new baseline benchmark which can:
  - Run on a large variation of machines
  - Can be use to project performance to larger machines and problems
  - Act as a comparison across generations of machines
  - Can be used to determine the effectiveness of multi-core processors

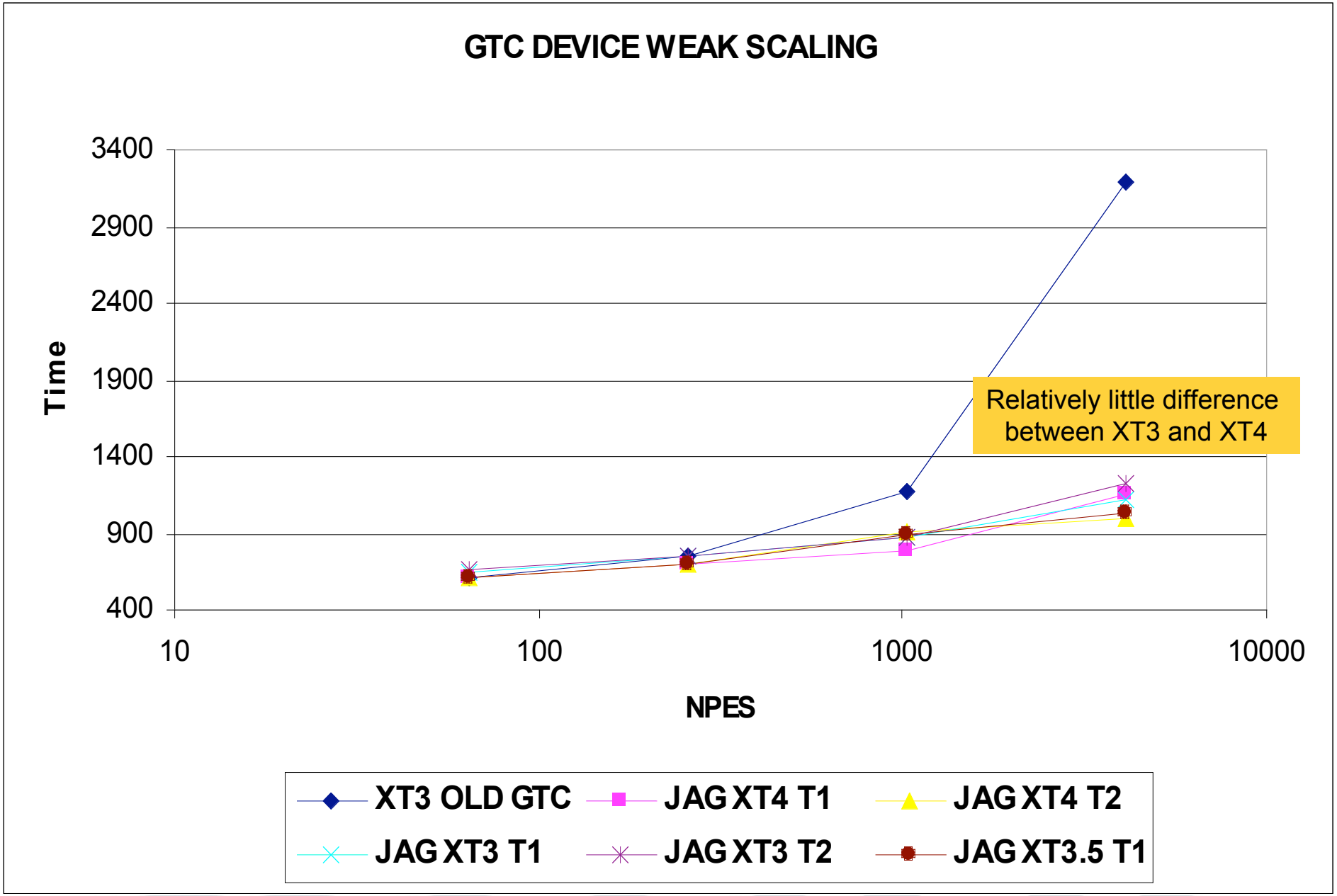
New weak scaling benchmark

- Scales the problem in both reactor size and # of particles
- Runs on 64 to 16384 processors with a step size of 4x
- Assumes a constant decomposition of 64 slices in direction of the torus
- Reaches a reactor size the size of ITER at 16384

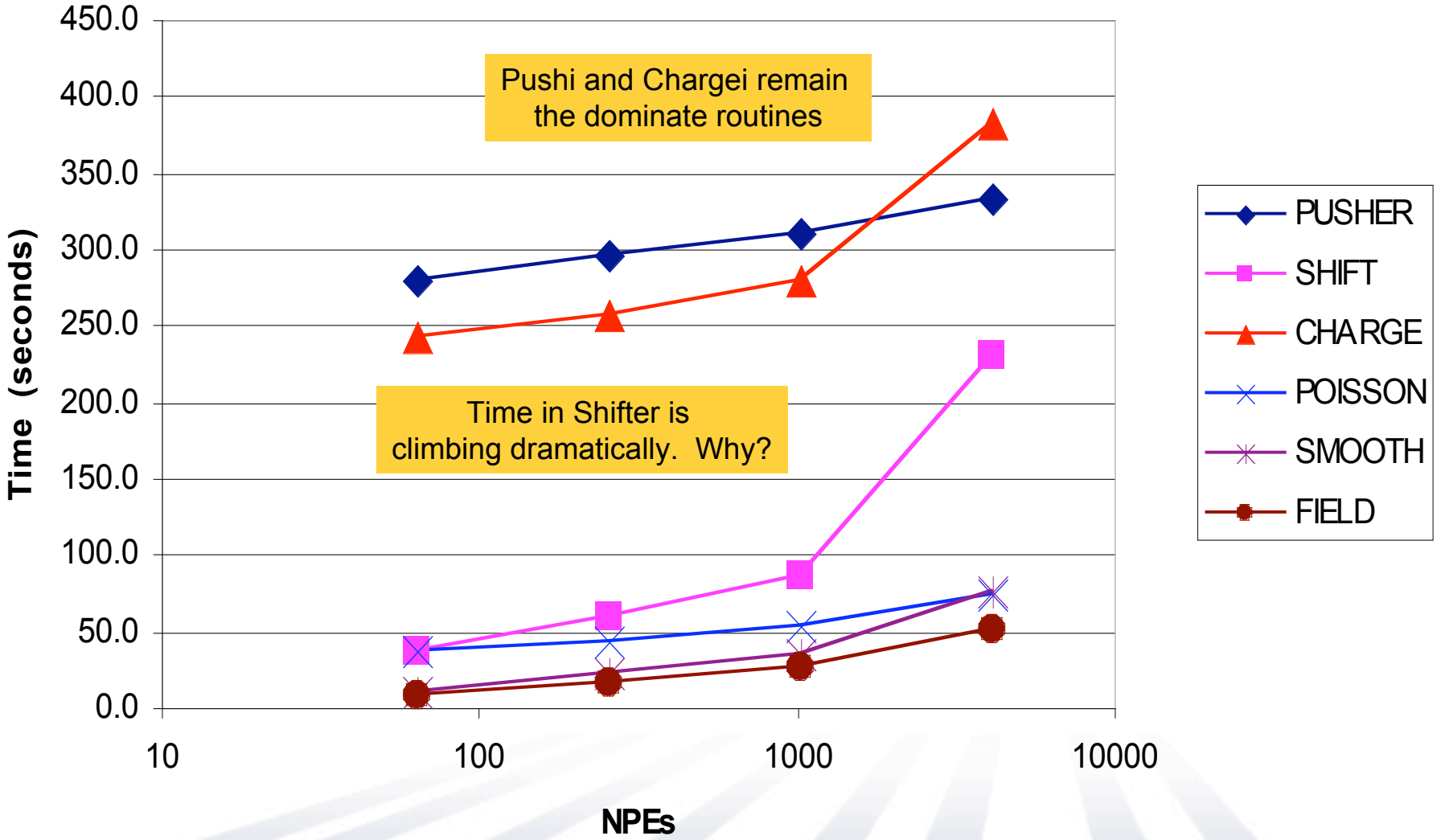
Data Collected

- Scaling data from 64 to 16384 processors on XT3/XT4 jaguar at ORNL, compare to old GTC
- Examine differences of running on either only the XT4 or XT3, or “whatever you get”
- Examine the scaling of each component
- Examine the data of dual core vs single core

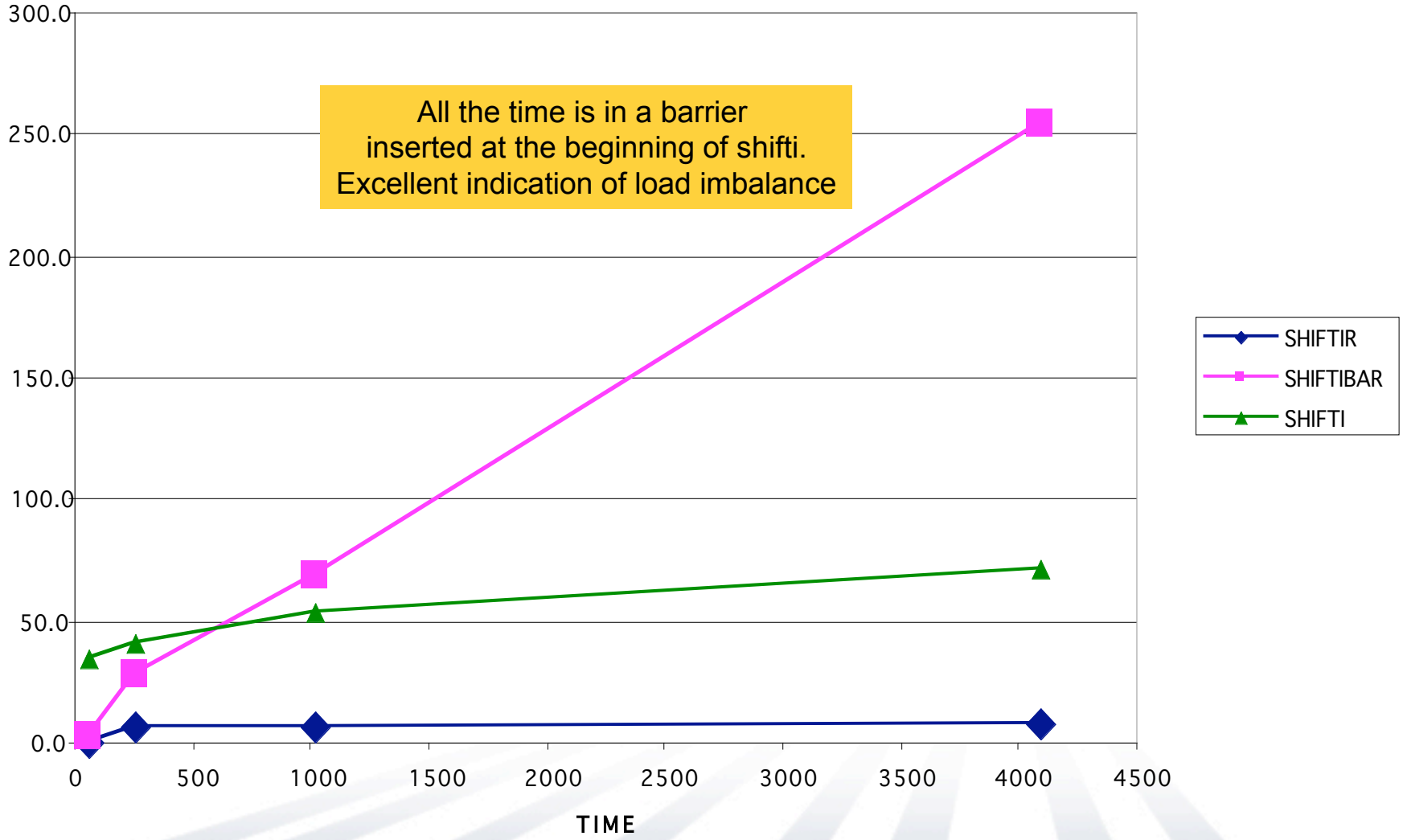

GTC DEVICE WEAK SCALING
Old GTC code does not scale well when increasing the device size
Scaling of new code is ok but is not as flat as we like
TIME
3500
3000
2500
2000
1500
1000
500
0
64
256
1024
4096
16384
NPES
XT3 OLD GTC
JAG XT4 T1
JAG XT4 T2
JAG XT3 T1
JAG XT3 T2
JAG XT3.5 T1


GTC DEVICE WEAK SCALING
Time
3400
2900
2400
1900
1400
900
400
10
100
1000
10000
NPES
Relatively little difference between XT3 and XT4
XT3 OLD GTC
JAG XT4 T1
JAG XT4 T2
JAG XT3 T1
JAG XT3 T2
JAG XT3.5 T1

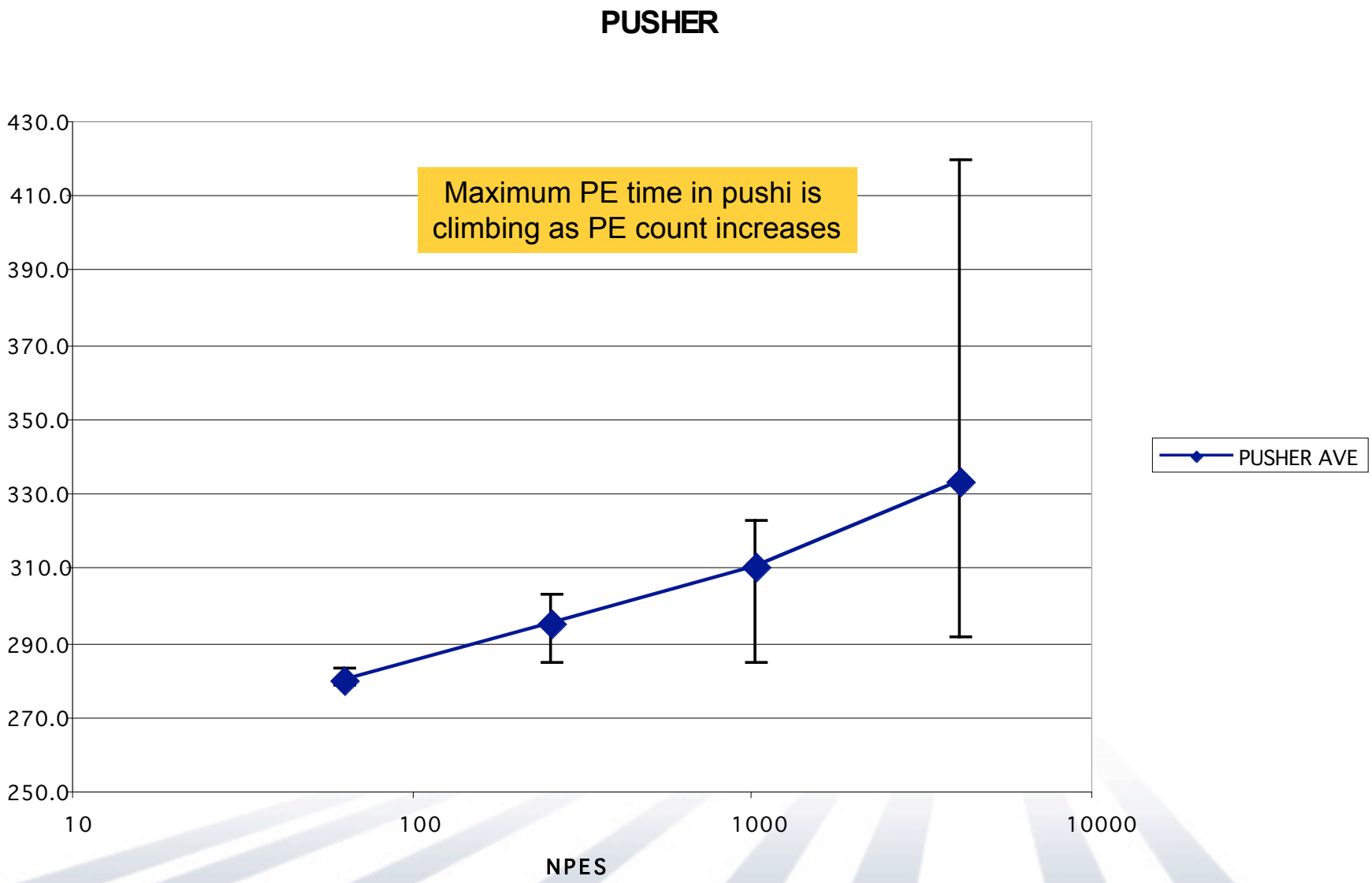
Component Times



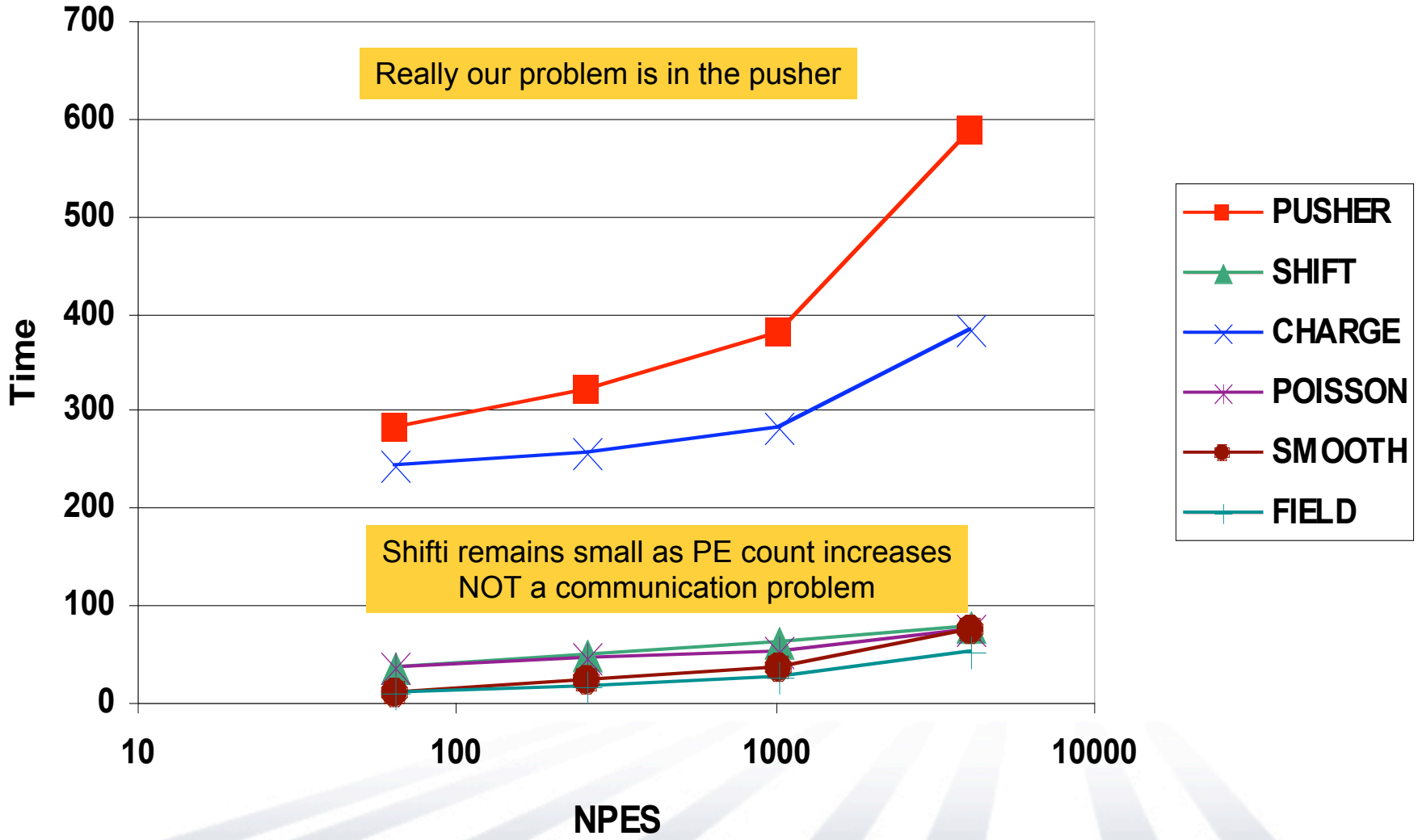
Phases of SHIFT



Plot of Max, Average, and Min time per PE in Pushi



Component Times with Barrier in Pushi

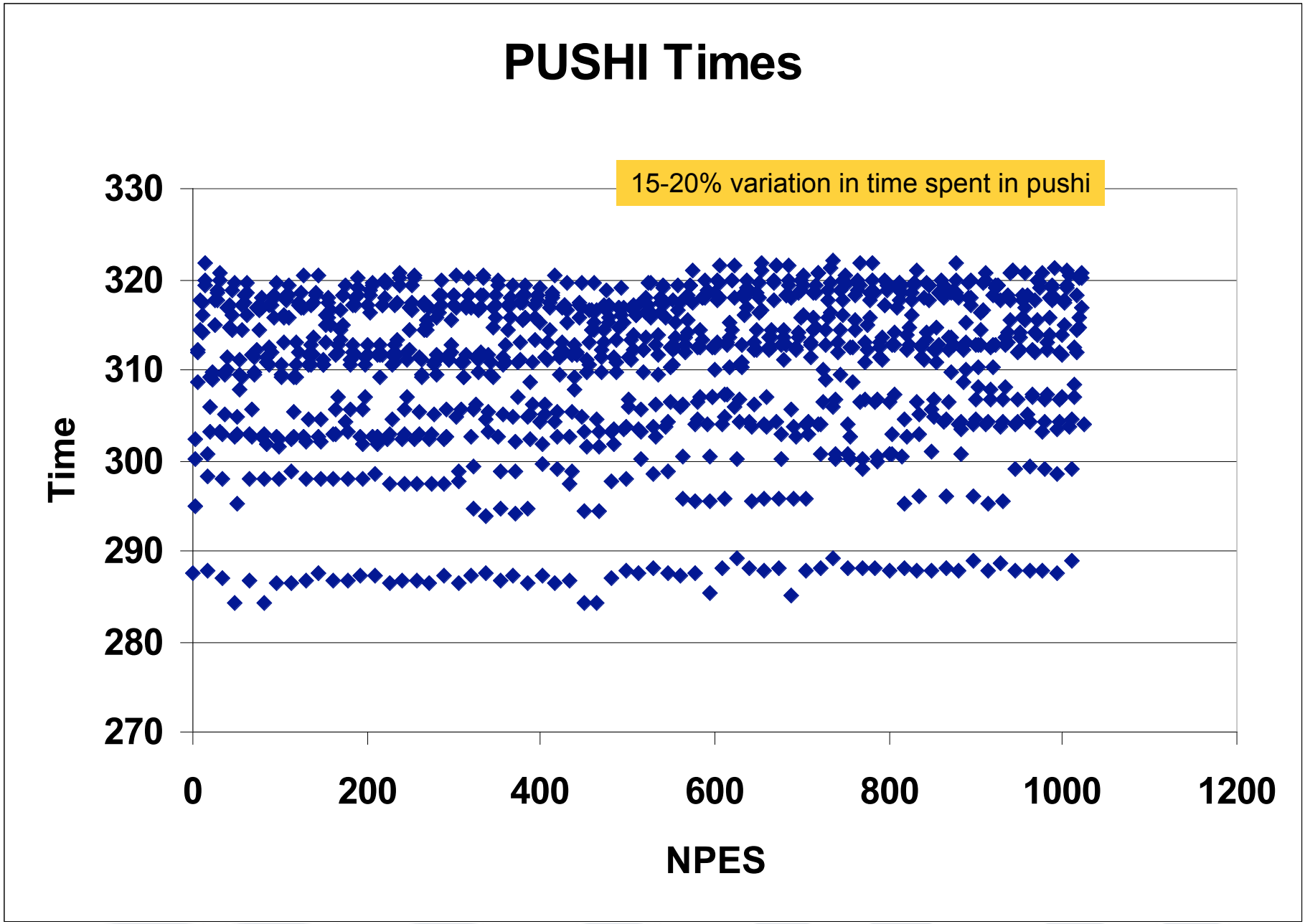


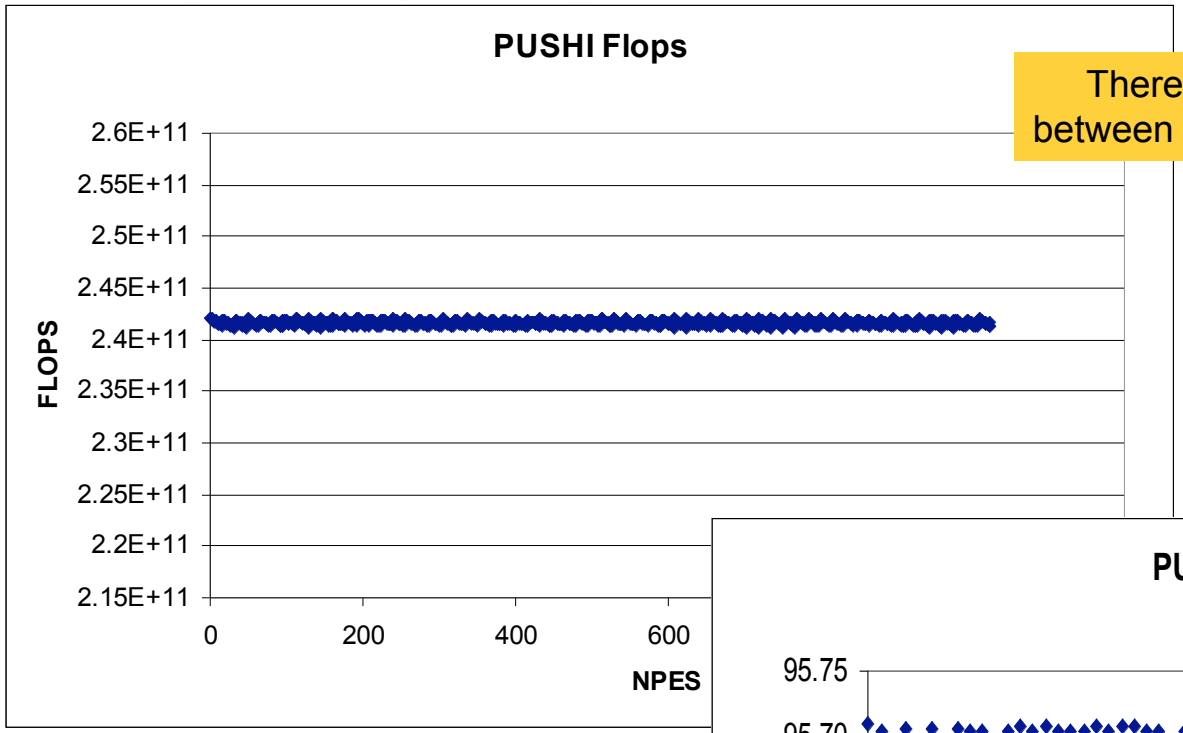
Instrument PUSHI using pat_region_begin

- Wanted to collect more data on pushi, so we instrumented the program using Craypat regions

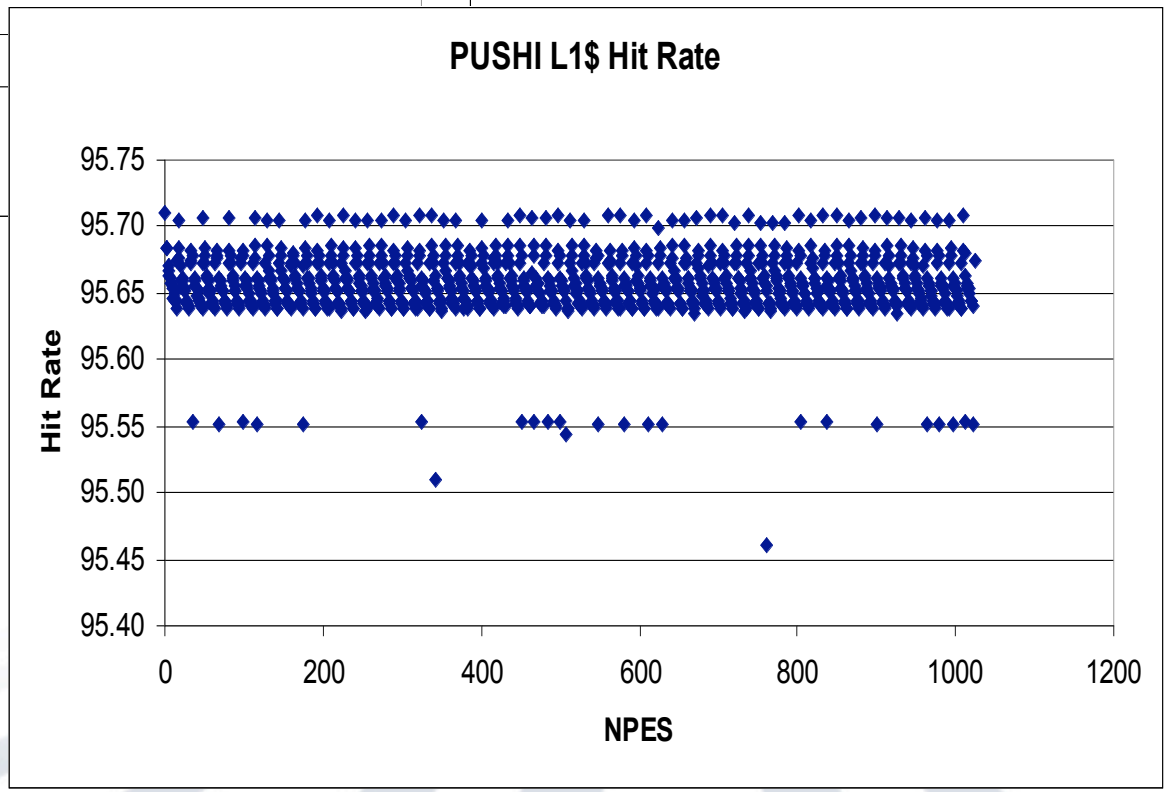
```
include “pat_apif.h”
call PAT_region_begin( 10, "pushi" ,istat )
call pushi(…)
call PAT_region_end( 10,istat )
```

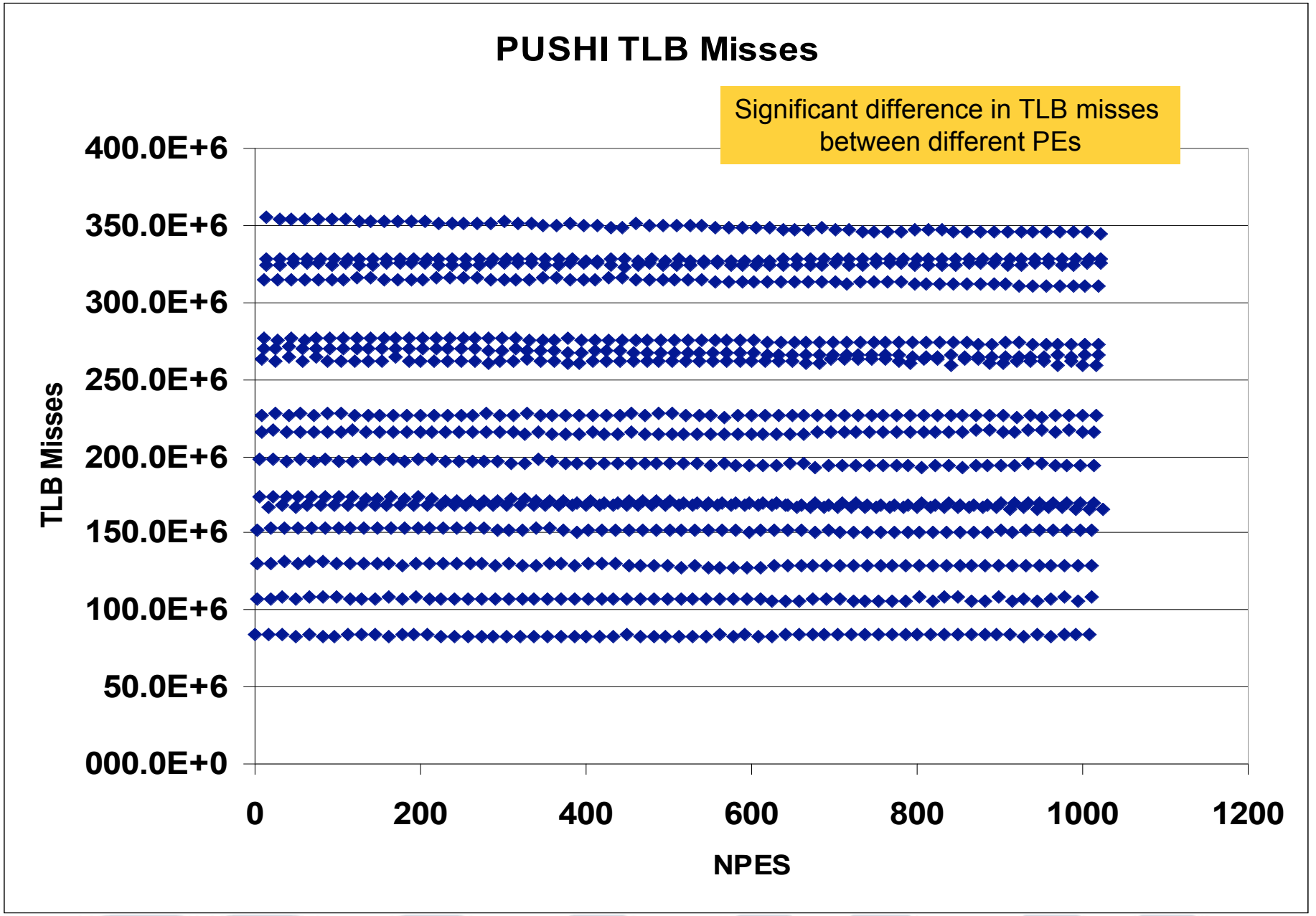
Compile with craypat module loaded
Relink using pat_build
Ran the program
Ran pat_report on resulting .xf file.

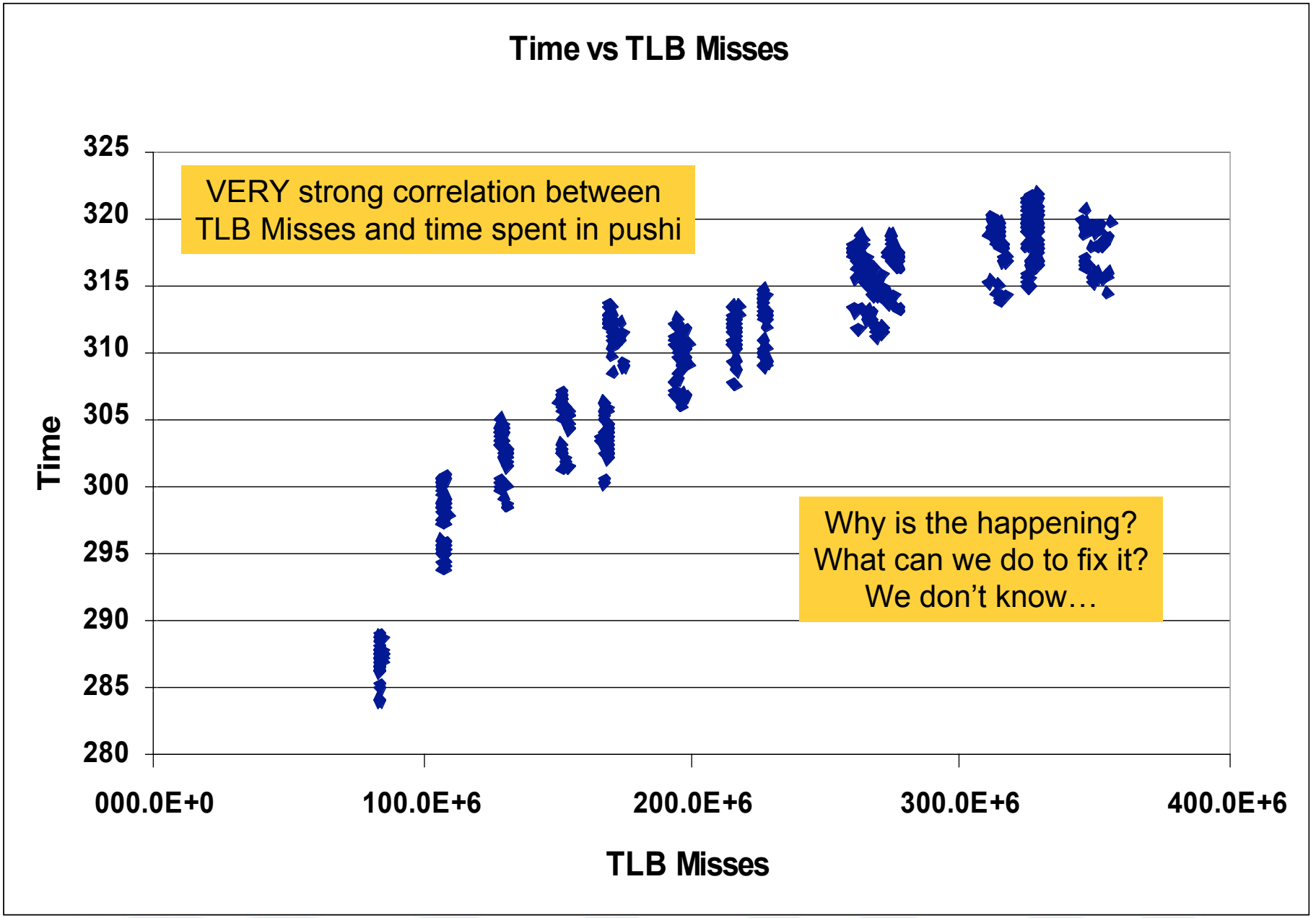

PUSHI Times
15-20% variation in time spent in pushi
330
320
310
300
290
280
270
Time
0
200
400
600
800
1000
1200
NPES



There does not appear to be a correlation between time and either FLOPs or cache hit rate




PUSHI TLB Misses
Significant difference in TLB misses between different PEs
400.0E+6
350.0E+6
300.0E+6
250.0E+6
200.0E+6
150.0E+6
100.0E+6
50.0E+6
000.0E+0
TLB Misses
0
200
400
600
800
1000
1200
NPES


Time vs TLB Misses
VERY strong correlation between TLB Misses and time spent in pushi
Why is the happening?
What can we do to fix it?
We don't know…
Time
325
320
315
310
305
300
295
290
285
280
000.0E+0
100.0E+6
200.0E+6
300.0E+6
400.0E+6
TLB Misses

How can we improve the effectiveness of the TLBs?

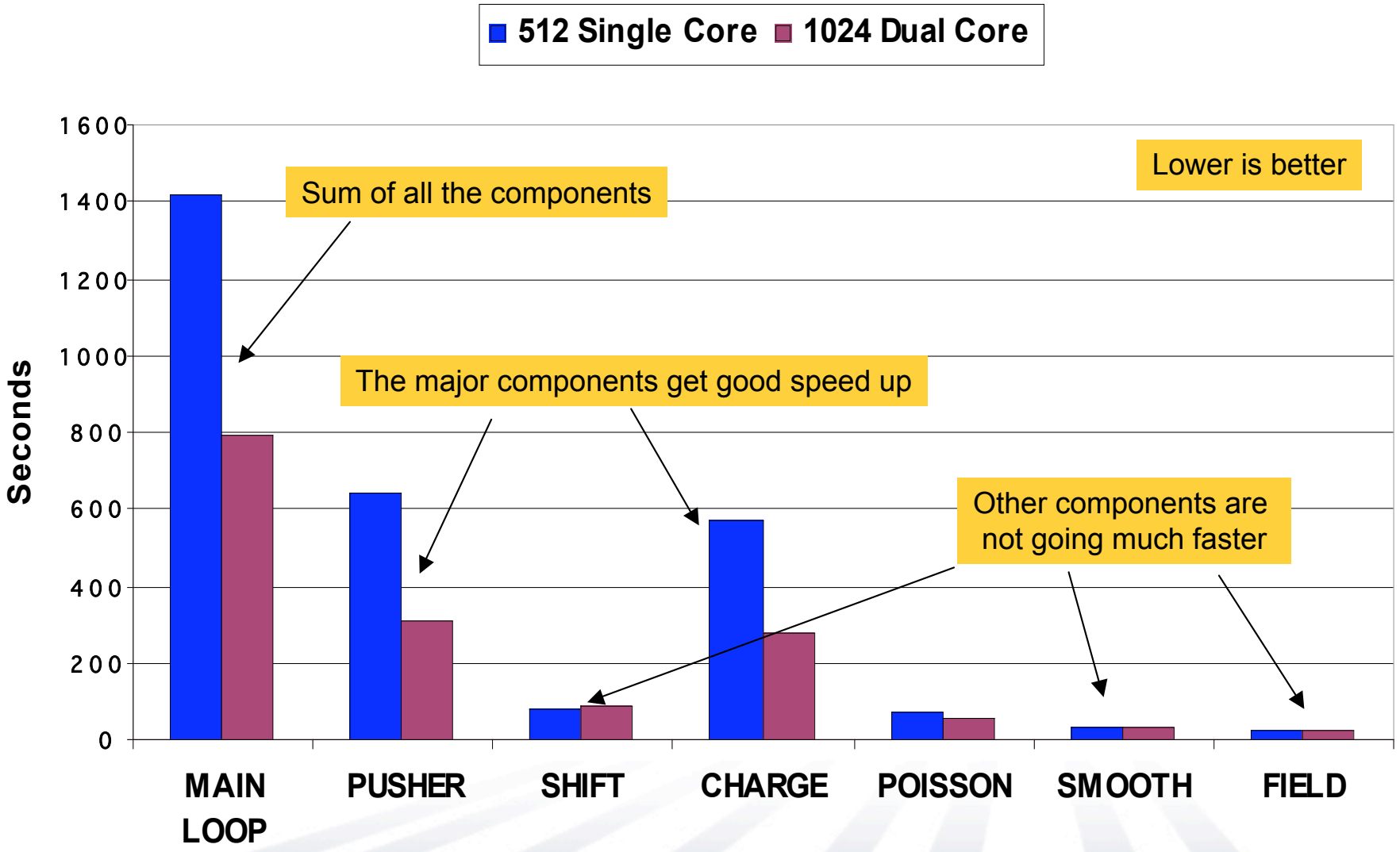
- If we use 32 bit floats, will we have fewer TLB misses?
- Can we find the memory access pattern that is causing the TLB misses?
  - If we find it, can we change it?
- Will the AMD quad core perform better?
  - Even if it does, we are not really willing to wait.

Multi Core vs Single Core

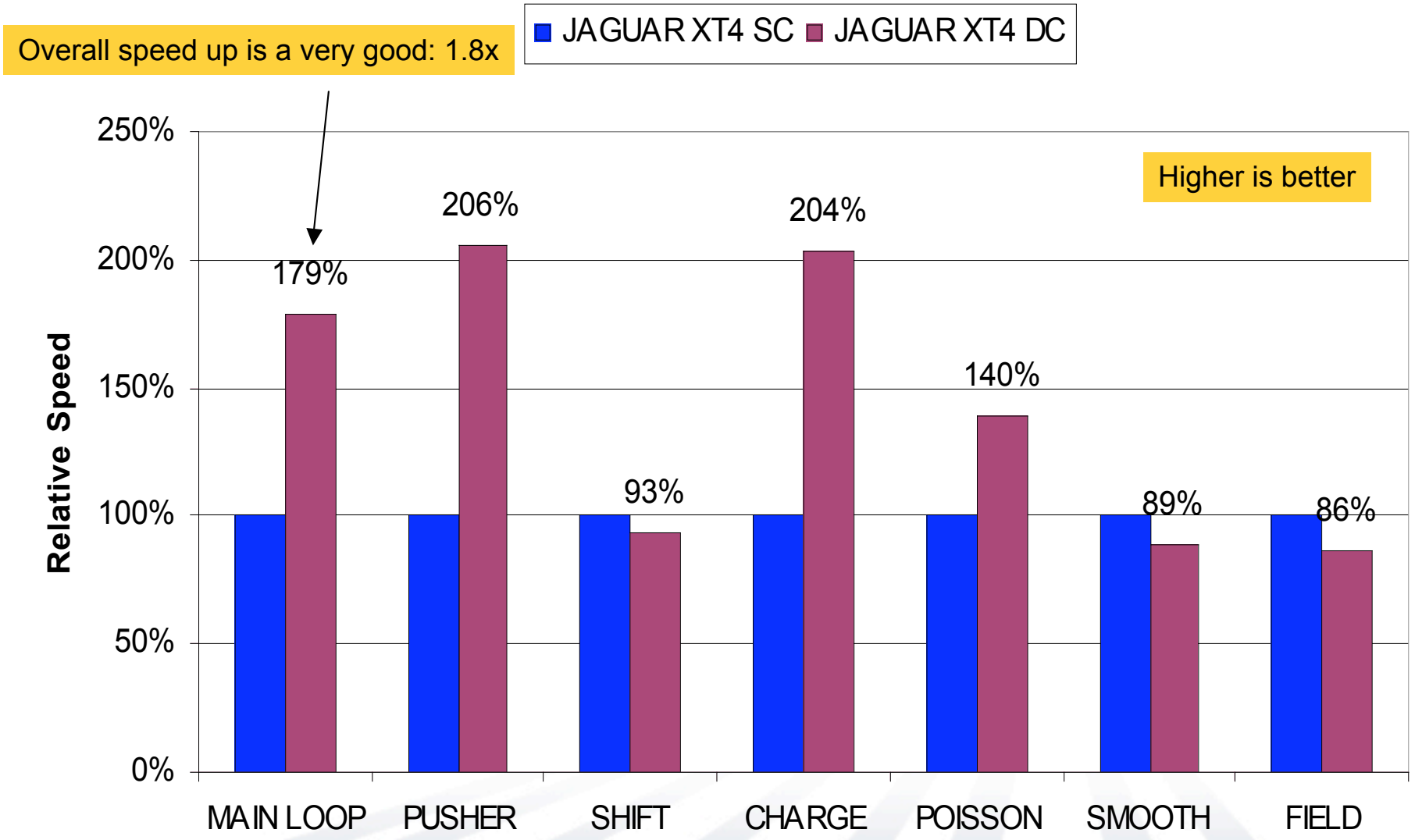
- Number of cores per socket are increasing
  - How does this effect GTC performance?
  - Will the profile change in the future?
  - Do we need to be worrying about different parts of the code?

- Designed a experiment were we ran a fixed problem size on 512 sockets using only a single core per socket, then ran the same problem using 1024 cores on 512 sockets
  - Effectively Strong Scaling from Single Core to Dual Core
  - Askes the question: How much faster will I get my science done given more cores?
  - Includes effects of algorithm scaling

Component Times using 512 XT4 sockets



Relative DC vs SC performance for all components



Future Work

- Explain and eliminate load imbalance inside of Pushi
  - Are TLBs really the problem? How do we fix it?
- Can we switch back to using 32 bit precision for all of some of the code?
- Examine PETSc solver for potential performance improvements
- Examine Shifti in more detail
- Reexamine the use of OMP inside of GTC
- Perform a weak scaling study where only the mesh and the number of particles are increased, but the device size does not change
- Try to project “how God would simulate ITER”

Conclusions

- New version of GTC provides a substantial increase in both capabilities and performance
  - Can run larger problems than the older version
  - New solver allows new science to be studied
  - Performance substantially better than the older version for large devices
- Scaling is good to 16K cores but more study is needed
  - Scaling should improve if we can eliminate load imbalance
- GTC can effectively use dual core Opterons
  - Main computational kernels see almost perfect speed up
  - Shifter and other kernels may become more important as the number of cores increases