Analysis and Optimization of Yee_Bench using Hardware Performance Counters

Ulf Andersson, Philip J. Mucci

Center for Parallel Computers (PDC), Royal Institute of Technology (KTH), Stockholm, Sweden
Innovative Computing Laboratory, UT Knoxville

ulfa at nada.kth.se, mucci at cs.utk.edu

September 13th, 2005
ParCo 2005, Malaga, Spain

Parallelldatorcentrum

Center for Parallel Computers (PDC)

- The biggest of the centers in Sweden that provides HPC resources to the scientific community. (~2000 procs, ~8TF)
  - Vastly different user bases, from Bio-informatics to CFD to CCM. Open to all Swedish academic institutions.
  - Multiple architectures, Linux the dominating OS.
    - IA64, EM64T, Pentium III/IV, Power 3

Yee_bench

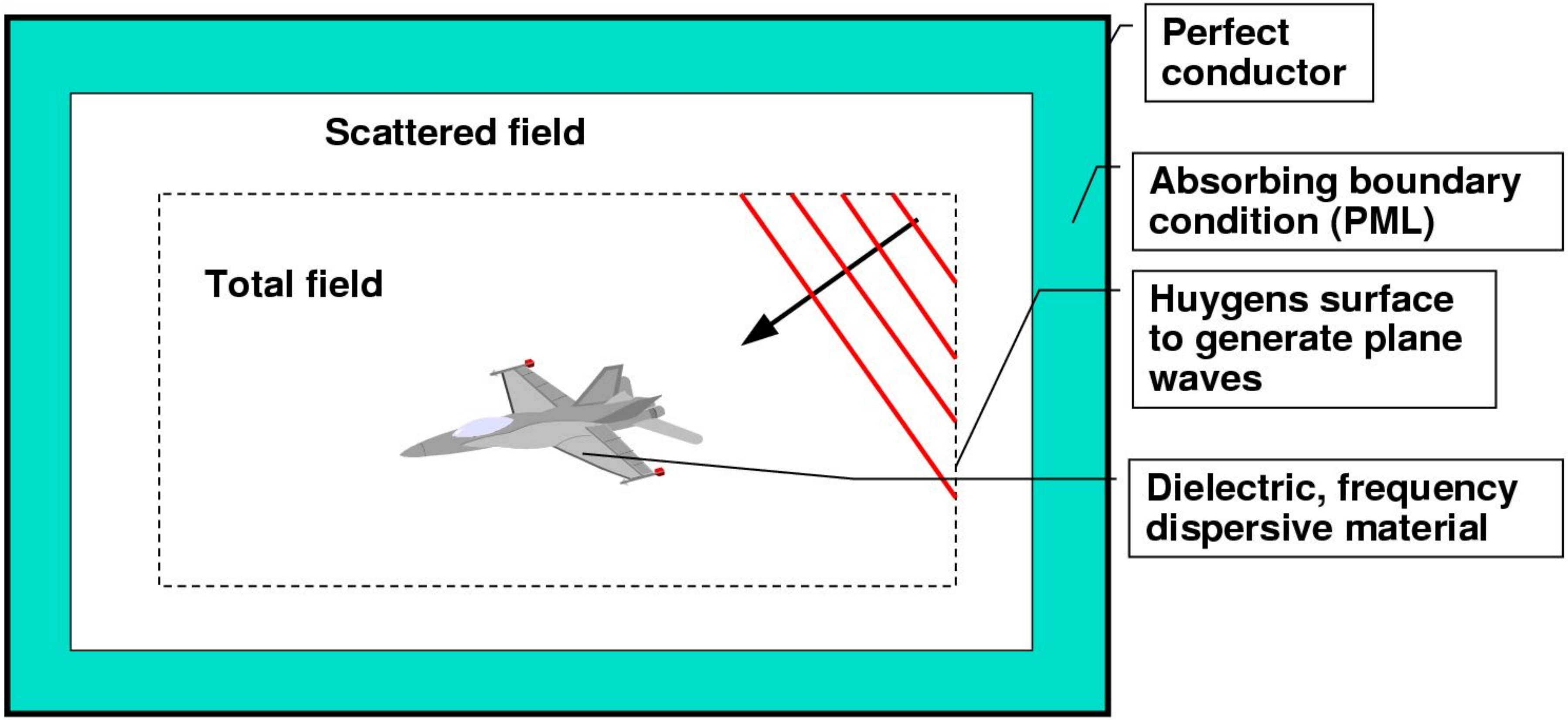
- A PDC developed benchmark
- Extensively used for architecture evaluation when purchasing new hardware
- Implements the core of the FDTD method in Computational Electromagnetics (CEM)
- Memory bandwidth bound
- 64-bit precision Fortran 90 version used here

Other codes used in the eval. process

- gromacs
- lapw1c (user code, eigenvalues)
- GemsTD (CEM user code)
- Gaussian
- Dalton
- DFT user code (DFT=density functional theory)
- EDGE (CFD user code)

The full FDTD method

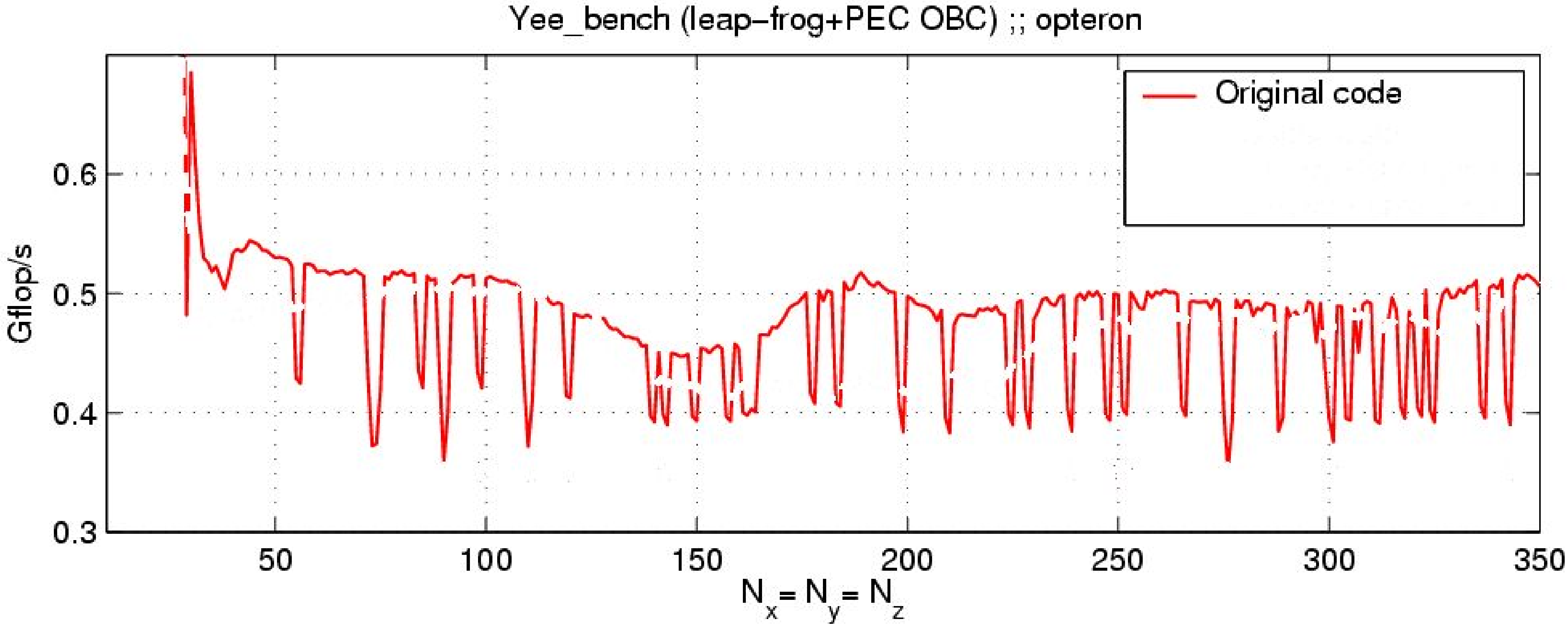
KTH YEE Finite Difference Time Domain (FDTD) code for Maxwell's equations in 2D and 3D



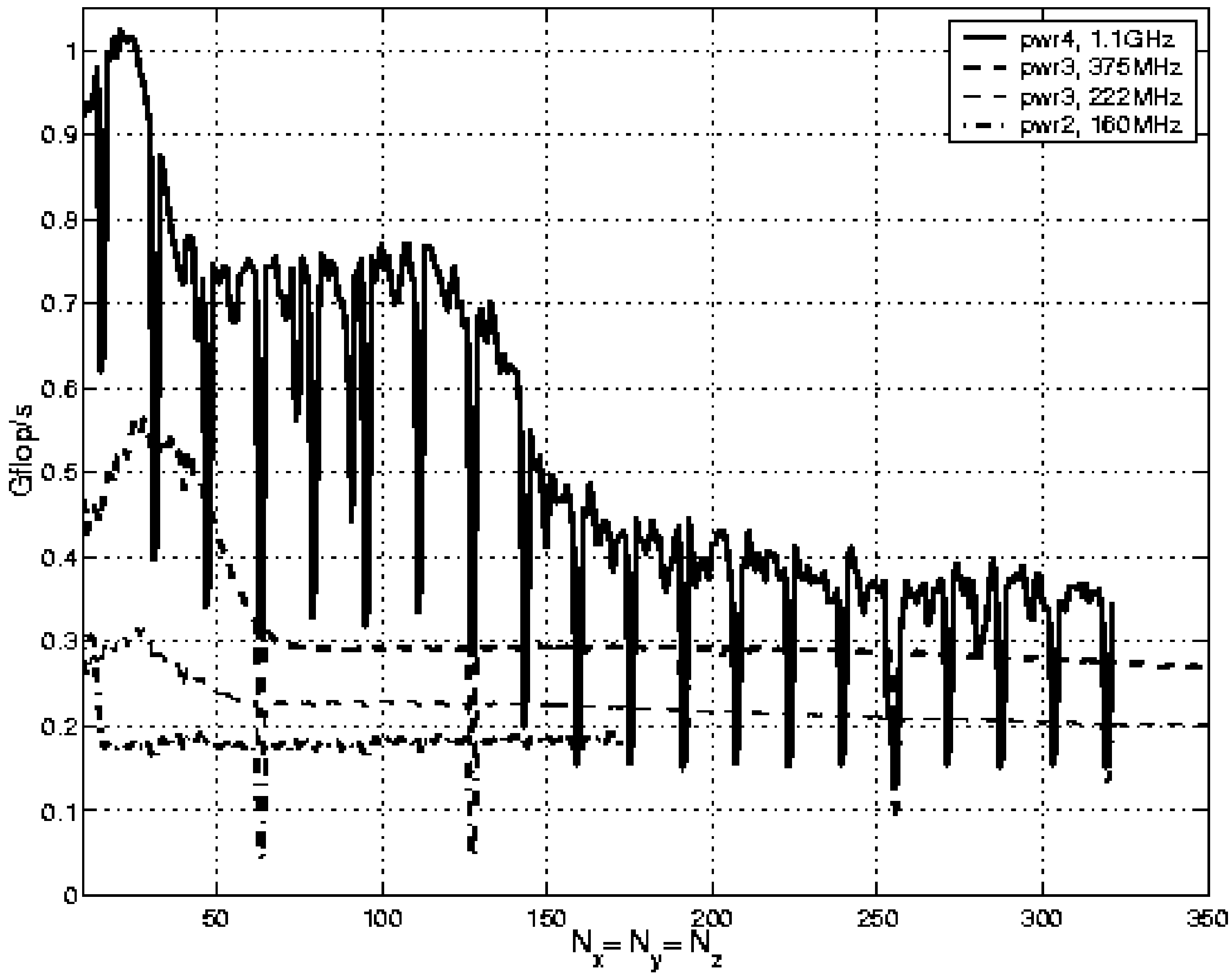
Opteron processor 846

- One CPU used from a four-way Opteron 846
- 2.0 GHz
- 8 Gbyte RAM (DDR 333)
- L1 cache is 64k, two-way set associative and cache line length is 64 bytes
- L2 cache is 1M, 16-way associative
- Results valid for both pgf90 and pathf90, and on all Opteron systems tested. OS is Linux.

Performance on an AMD Opteron



IBM results



 son

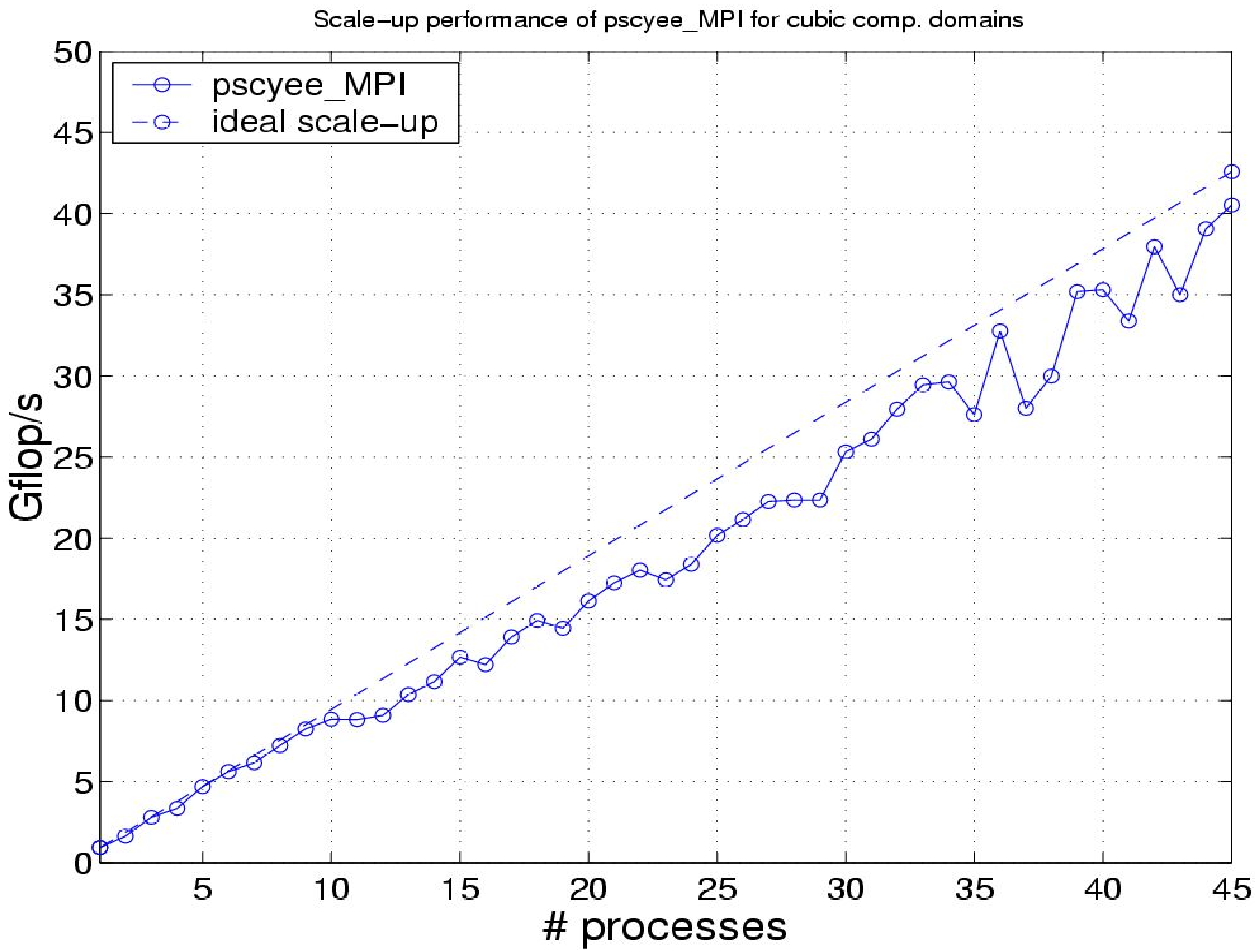
Parallel Performance

"The single most important impediment to good parallel performance is *still* poor single-node performance."

- William Gropp

Argonne National Lab

Parallel version of Yee_Bench on Itanium



Hardware Performance Counters

- Performance Counters are hardware registers dedicated to counting certain types of events within the processor or system.
  - Usually a small number of these registers (2,4,8)
  - Sometimes they can count a lot of events or just a few
  - Symmetric or asymmetric
  - May be on or off chip
- Each register has an associated control register that tells it what to count and how to do it. For example:
  - Interrupt on programmable counter overflow: IP sampling
  - User, kernel, interrupt mode

Availability of Performance Counters

- Most high performance processors include hardware performance counters.
  - AMD
  - Alpha
  - Cray MSP/SSP
  - PowerPC
  - Itanium
  - Pentium
  - MIPS
  - Sparc
  - And many others...

Available Performance Data

- Cycle count
- Instruction count
  - All instructions
  - Floating point
  - Integer
  - Load/store
- Branches
  - Taken / not taken
  - Mispredictions
- Pipeline stalls due to
  - Memory subsystem
  - Resource conflicts
- Cache
  - I/D cache misses for different levels
  - Invalidations
- TLB
  - Misses
  - Invalidations

PAPI

- **P**erformance **A**pplication **P**rogramming **I**nterface
- The purpose of PAPI is to implement a standardized portable and efficient API to access the hardware performance monitor counters found on most modern microprocessors.
- The goal of PAPI is to facilitate the optimization of parallel and serial code performance by encouraging the development of cross-platform optimization tools.
  - TAU: Instrumentation and Tracing
  - Kojak: Automated Bottleneck Analysis
  - HPCToolkit: Statistical Profiling

Hardware Performance Counter Virtualization by the OS

- Every process/thread appears to have its own counters.
- OS accumulates counts into 64-bit quantities for each thread and process.
  - Saved and restored lazily on context switch.
- All counting modes are supported (user, kernel and others).
  - Aggregate “caliper” type counting
  - IP sampling: histograms based on counter overflow.
- Counts are largely independent of load.

Data Collection with PAPIEX

- PapiEx: a command line tool that collects performance metrics along with PAPI data for each thread and process of an application.
  - No recompilation required.
- Based on PAPI and Monitor libraries.
- Uses library preloading to insert the instrumentation libraries before the application gets started. (via Monitor)
  - Does not work on statically linked or SUID binaries.

Some PapiEx Features

- Automatically detects multi-threaded executables.
- Supports PAPI counter multiplexing; use more counters than available hardware provides.
- Full memory usage information.
- Simple instrumentation API.
  - Called PapiEx Calipers.

PapiEx Sample Output

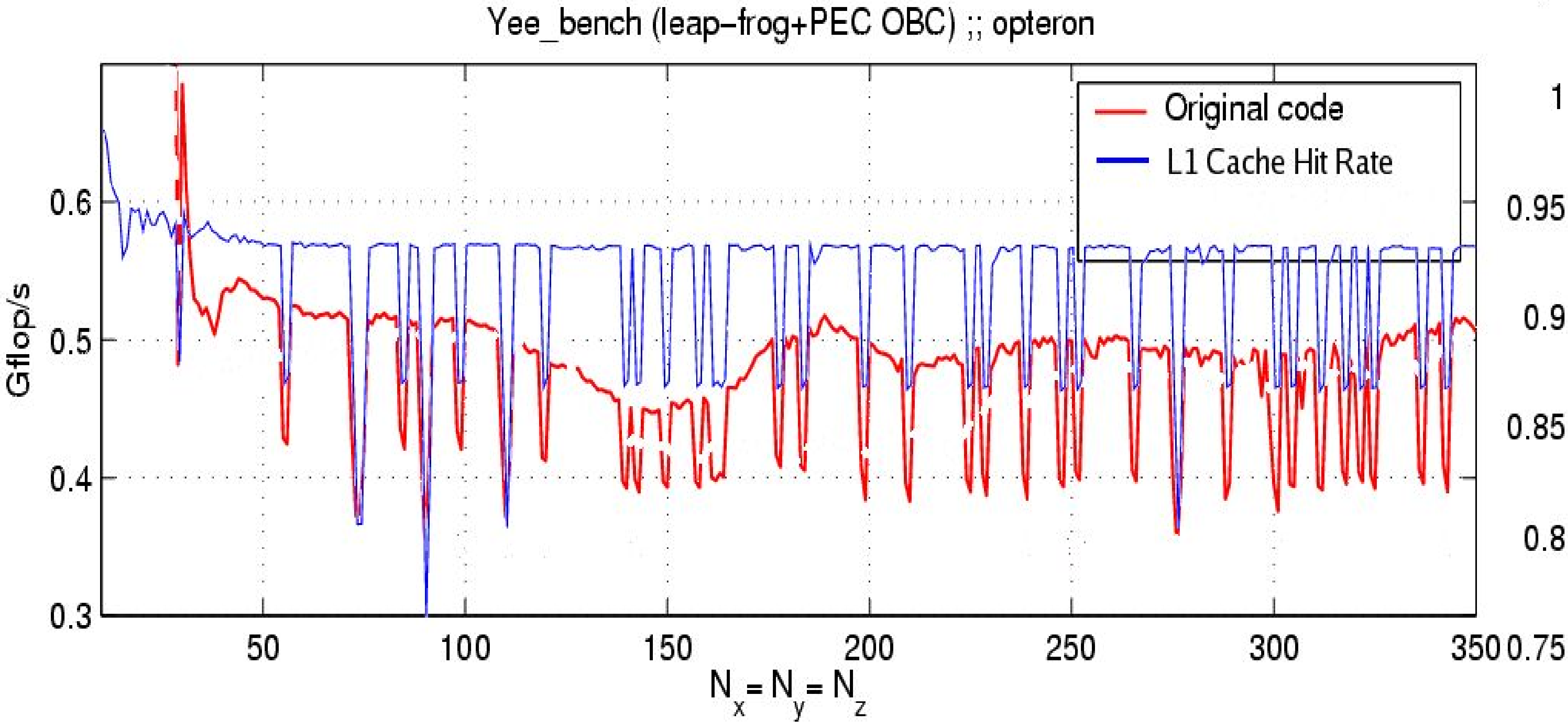
```
PapiEx Version:          0.99rc2
Executable:              /afs/pdc.kth.se/home/m/mucci/summer/a.out
Processor:               Itanium 2
Clockrate:               900.000000
Parent Process ID:       8632
Process ID:              8633
Hostname:                h05n05.pdc.kth.se
Options:                 MEMORY
Start:                   Wed Aug 24 14:34:18 2005
Finish:                  Wed Aug 24 14:34:19 2005
Domain:                  User
Real usecs:              1077497
Real cycles:             969742309
Proc usecs:              970144
Proc cycles:             873129600
PAPI_TOT_CYC:            850136123
PAPI_FP_OPS:             40001767
Mem Size:                4064
Mem Resident:            2000
Mem Shared:              1504
Mem Text:                16
Mem Library:             2992
Mem Heap:                576
Mem Locked:              0
Mem Stack:               32

Event descriptions:
Event: PAPI_TOT_CYC
        Derived: No
        Short Description: Total cycles
        Long Description: Total cycles
        Developer's Notes:
Event: PAPI_FP_OPS
        Derived: No
        Short Description: FP operations
        Long Description: Floating point operations
        Developer's Notes:
```

Monitor

- Portable Linux library for transparently catching "important" events via LD_PRELOAD.
- Callbacks to a tool library on:
  - Process/Thread creation, destruction.
  - fork/exec/dlopen.
  - exit/_exit/Exit/abort/assert.
  - User can easily add any number of wrappers.

AMD perf. vs L1 cache hit rate



Yee_Bench Kernel

```
do k=1,nz          ! The magnetic field update
  do j=1,ny        ! Electric field update is very similar.
    do i=1,nx
      Hx(i,j,k) = Hx(i,j,k) +                              &
                  (  (Ey(i,j,k+1)-Ey(i,j  ,k))*Cbdz +      &
                     (Ez(i,j,k  )-Ez(i,j+1,k))*Cbdy  )
      Hy(i,j,k) = Hy(i,j,k) +                              &
                  (  (Ez(i+1,j,k)-Ez(i,j,k  ))*Cbdx +      &
                     (Ex(i  ,j,k)-Ex(i,j,k+1))*Cbdz  )
      Hz(i,j,k) = Hz(i,j,k) +                              &
                  (  (Ex(i,j+1,k)-Ex(i  ,j,k))*Cbdy +      &
                     (Ey(i,j  ,k)-Ey(i+1,j,k))*Cbdx  )
    end do
  end do
end do
```

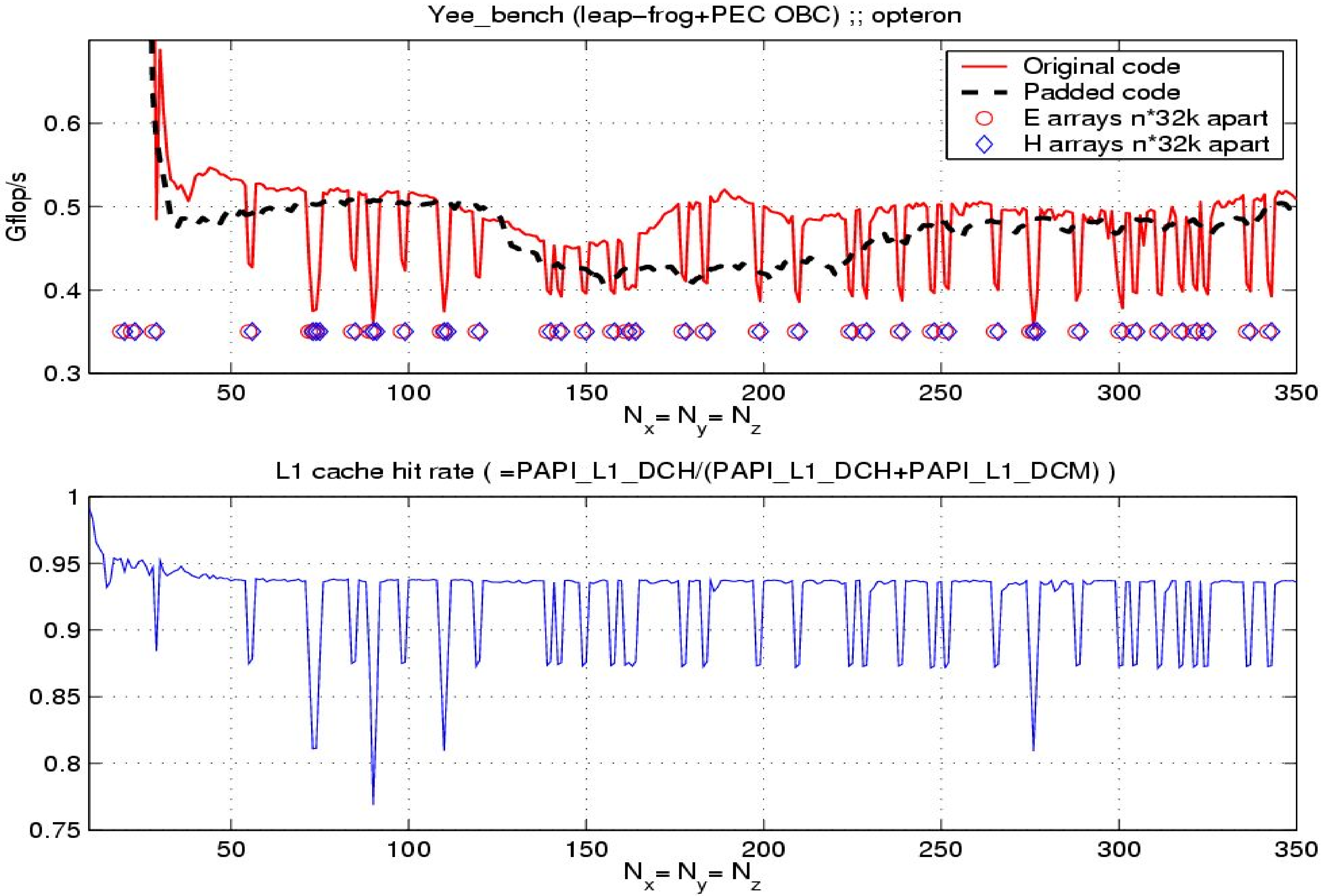
Yee_Bench allocations (improved)

```
Hx(1:nx  +padHx(1),1:ny  +padHx(2),1:nz  +padHx(3))
Hy(1:nx  +padHy(1),1:ny  +padHy(2),1:nz  +padHy(3))
Hz(1:nx  +padHz(1),1:ny  +padHz(2),1:nz  +padHz(3))
Ex(1:nx+1+padEx(1),1:ny+1+padEx(2),1:nz+1+padEx(3))
Ey(1:nx+1+padEy(1),1:ny+1+padEy(2),1:nz+1+padEy(3))
Ez(1:nx+1+padEz(1),1:ny+1+padEz(2),1:nz+1+padEz(3))
```

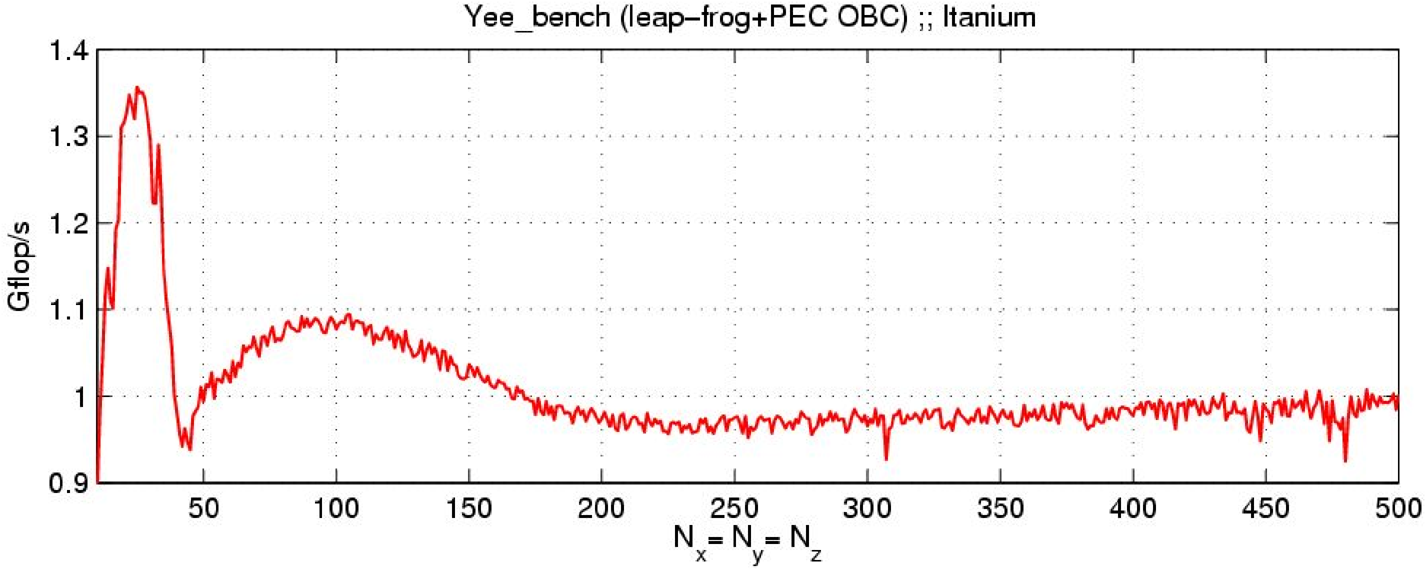
Allocation analysis

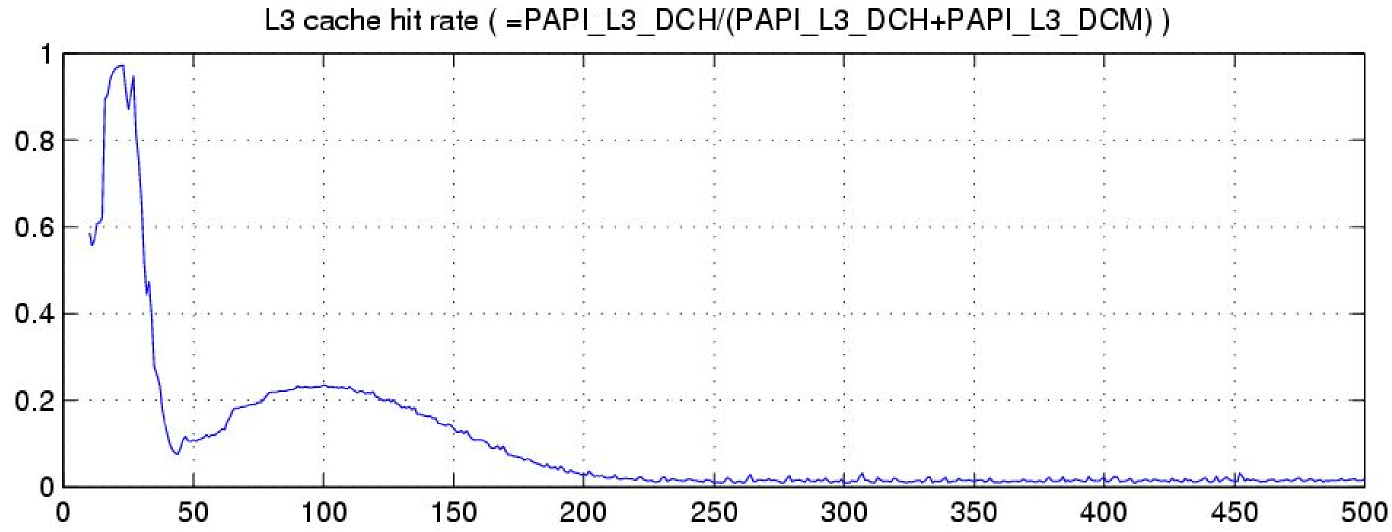
- LOC() shows that all arrays begin at the start of a page.
  - page size = 4k
  - L1 cache = 64k (two-way => 32k)
  - If arrays are allocated with a distance that is a multiple of 32k we get cache contention. (on average, this happens for approx. 12.5 % of the problem sizes)
- The fact that there are performance problems indicates that pages are physically contiguous.

AMD performance, new vs old

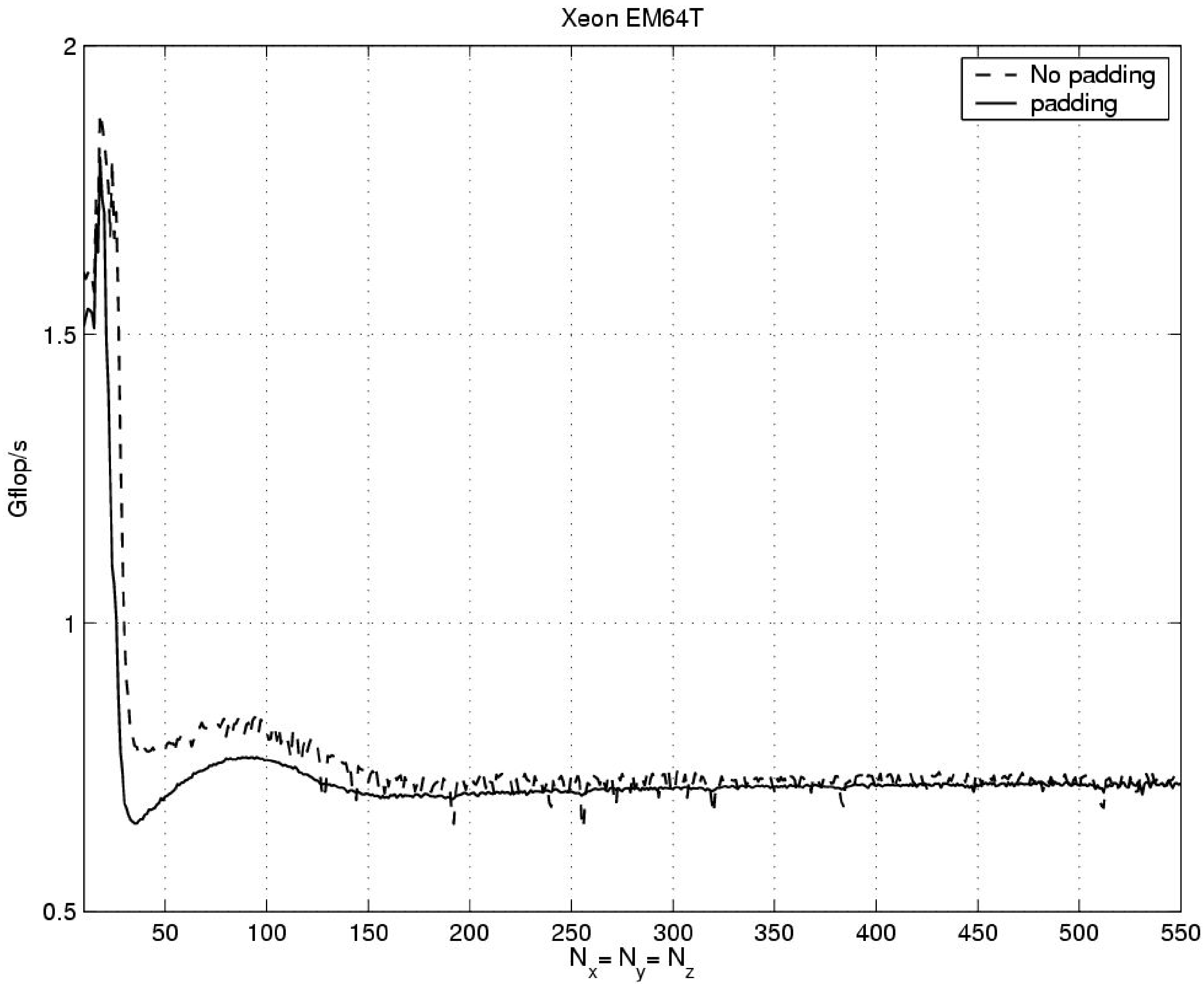


Itanium results





Nocona results



An improved allocate

- Cache specific allocator?
  - Suffers portability, complexity and interface issues.
- Cache aware allocator: (generic linesize)
  - Avoid returning allocations on powers of two for “large” allocations.
    - Pad by: **(linesize * (num_allocations MOD lines_per_page))**
    - Allocate an extra page (when necessary).
  - Assuming ALLOCATE() is built on brk()/mmap().
  - Fast, simple and portable.

Conclusion

- An easy to use performance monitor tool (papiex) was essential in order to understand the performance of a benchmark code (Yee_bench) on the AMD Opteron.
- Fortran 90 ALLOCATE() could easily be coded to avoid basic associativity conflicts.

Links

- http://icl.cs.utk.edu/~mucci/mucci_talks.html
- Software:
  - http://icl.cs.utk.edu/~mucci/monitor
  - http://icl.cs.utk.edu/~mucci/papiex
  - http://icl.cs.utk.edu/papi
  - Yee_bench available upon request (from ulfa)

Questions: ulfa at pdc.kth.se & mucci at cs.utk.edu