TECHNISCHE UNIVERSITÄT DRESDEN
Center for Information Services and High Performance Computing (ZIH)

Analysis and Optimization of a Molecular Dynamics Code using PAPI and the Vampir Toolchain

May 2, 2012

Thomas William
Zellescher Weg 12
Willers-Bau A 34
+49 351 - 463 32446

Thomas.William@ZIH.TU-Dresden.de

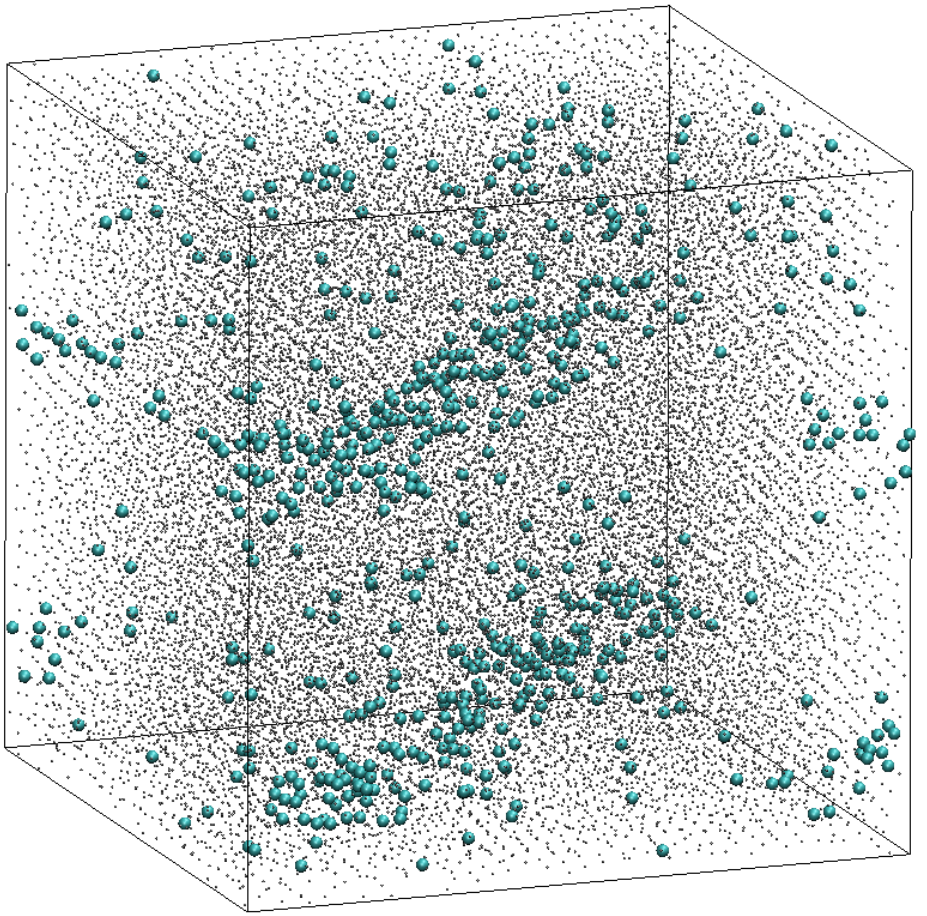
ZIH
Center for Information Services & High Performance Computing

Overview

Overview

Introduction

- IU
- ZIH
- FutureGrid
- MD



Introduction: MD Code

- Classical molecular dynamics simulation of dense nuclear matter consisting of either fully ionized atoms or free neutrons and protons
- Main targets are studies of the dense matter in white dwarf and neutron stars.
- Interaction potentials between particles treated as classical two-particle central potentials
  - No complicated particle geometry or orientation
- Electrons can be treated as a uniform background charge (not explicitly modeled)

MD Code Details

- Particle-particle-interactions have been implemented in a multitude of ways
- Located in different files PP01, PP02 and PP03
- The code blocks are selectable using preprocessor macros
  - **PP01** is the original implementation with no division into the Ax, Bx or NBS blocks
  - **PP02** implements the versions in use by the physicists today
    - 3 different implementations for the Ax block
    - 3 implementations for the Bx block
    - No manual blocking (NBS)
  - **PP03** includes new routines
    - 3 Ax blocks
    - 8 Bx blocks
    - Can be blocked using the NBS value

MD Code Details

- Two sections of code each have two or more variations
- One section is labelled `A` and the other `B`
- Variations are numbered
- MD can be compiled as a serial, OpenMP, MPI or MPI+OpenMP

```
make MDEF=XRay md_mpi ALG=PP02 BLKA=A0 BLKB=B2 NBS="NBSX=0"
```

MD Workflow

The structure of the algorithm is simple.

- First reads in a parameter file `runmd.in`
- And an initial particle configuration file `md.in`
- Program then calculates the initial set of accelerations
- Enters a time-stepping loop, a triply nested ”do” loop

Runtime Parameters

```
!Parameters:
   sim_type ='ion-mixture',  !simulation type
   tstart   =      0.00,     !start time
   dt       =     25.00,     !time step (fm/c)
!Warmup:
   nwgroup  =         2,     !groups
   nwsteps  =        50,     !steps per group
!Measurement:
   ngroup   =         2,     !groups
   ntot     =         2,     !per group
   nind     =        25,     !steps between
   tnormalize =      50,     !temp normal.
   ncom     =        50,     !center-of-mass
                              !motion cancel.
```

Figure : Runtime parameters, snippet from `runmd.in`

The Main Loop

```
      do 100 ig=1,ngroup
         initialize group ig statistics
        do 40 j=1,ntot
           do i=1,nind
              !computes forces
              !updates x and v
              call newton
           enddo
           call vtot
        enddo
        compute group ig statistics
 40     continue
100  continue
```

Figure : Simplified version of the main loop

MD Implementation Details - `newton` module

- Forces are calculated in the `newton` module in a pair of nested do-loops
  - Outer loop goes over target particles
  - Inner loop over source particles
- Targets are assigned to MPI processes in a round-robin fashion
- Within each MPI process, the work is shared among OpenMP threads

Overview

2 Serial Analysis

XRay - a Cray XT5m™

- Cray XT5m provided by the FutureGrid project
- XT5m is a 2D mesh of nodes
- Each node has two sockets each having four cores
- AMD Opteron 23 ”Shanghai” (45mm) running at 2.4 GHz
- 84 compute nodes with a total of 672 cores
- pgi/9.0.4 using the XT PE driver `xtpe-shanghai`

Time Constraints

- 5k particles measurement takes 5 minutes
- 27k particles measurement takes 1 hour
- 55k particles measurement takes 10 hours

Overview PP01, PP02, and PP03

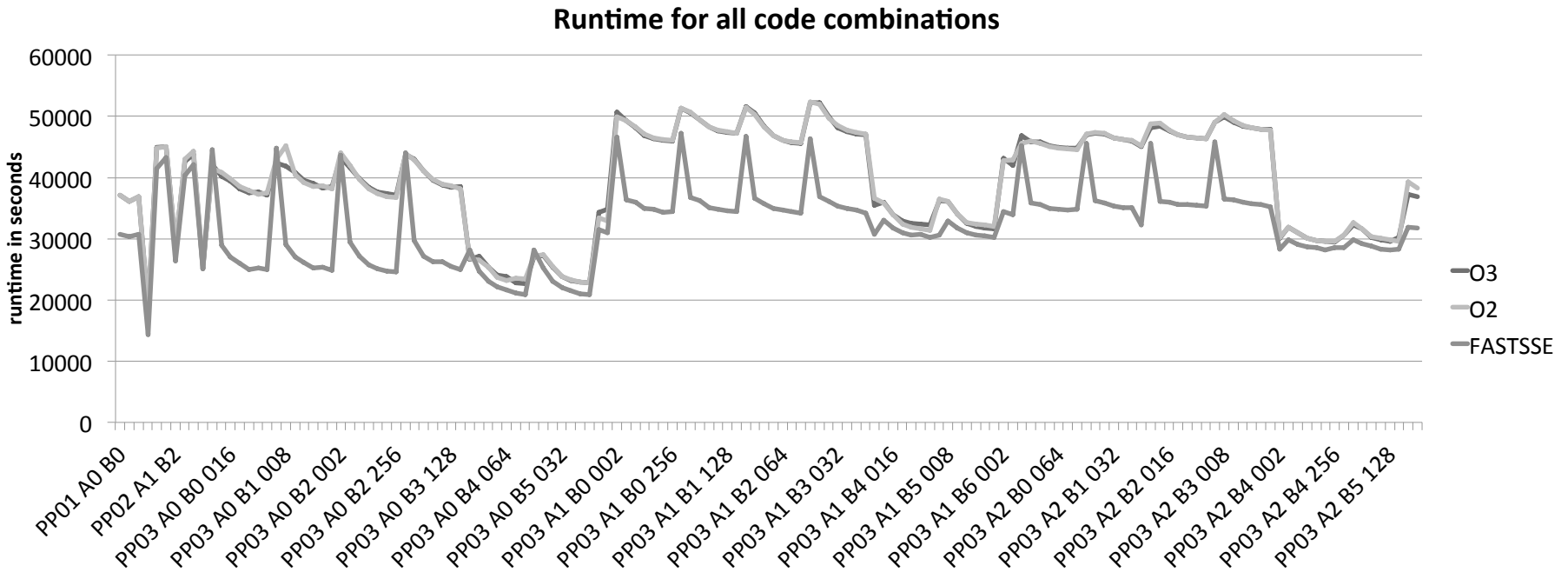
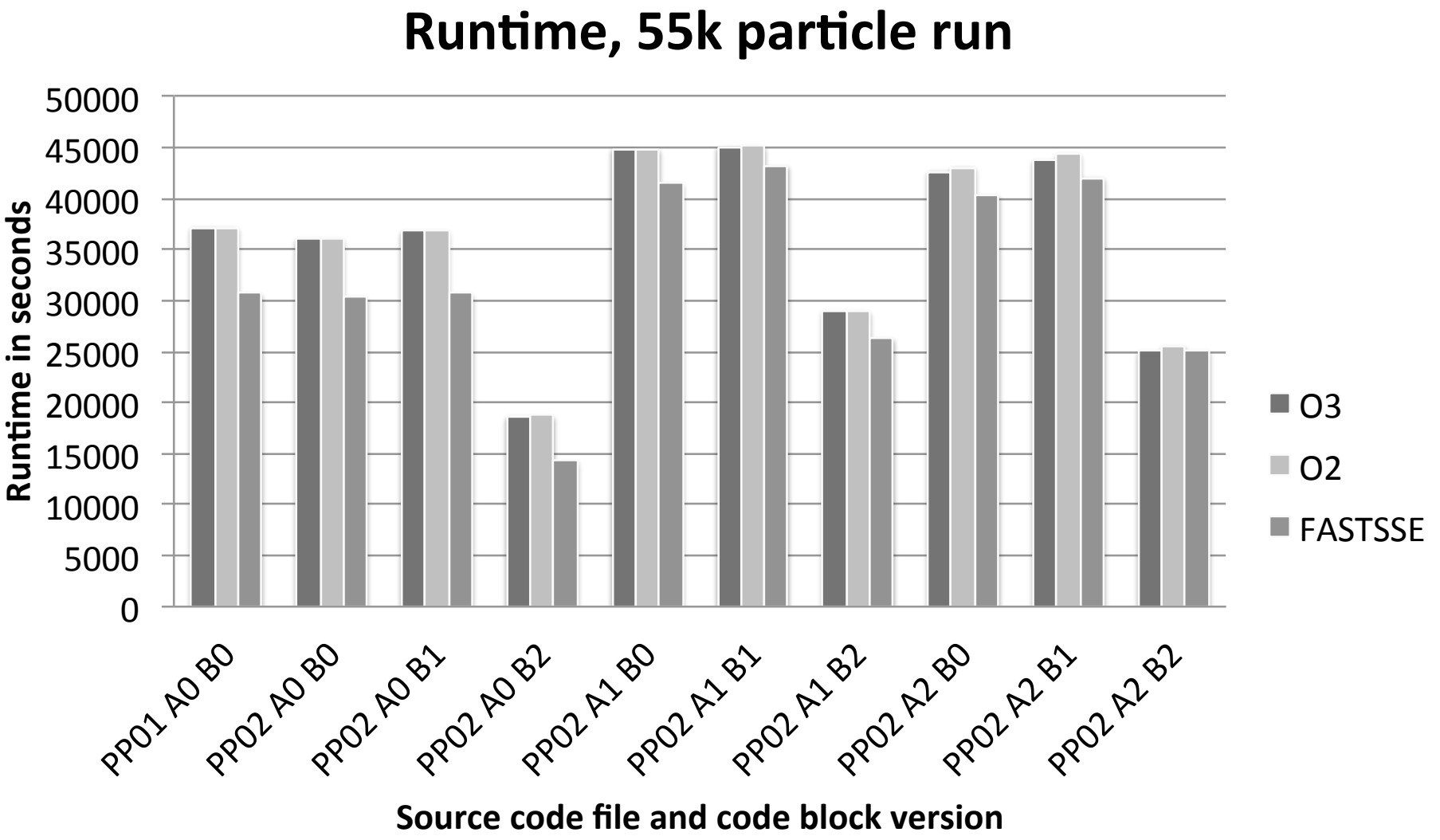


Figure : Overview of the runtimes for all (143) code-block combinations with an ion-mix input dataset of 55k particles. The naming scheme for the measurements is ”source-code-file A-block B-block blockingfactor”

PP02 Code Blocks



Annotated Source for -O3

```
##             do 90 j=i+1,n-1
##             !------ Block A ------
## #if defined(A0)
##                 r2=0.0d0
   movsd    %xmm2, %xmm1
   movq     %r12, %rdx
   movq     %r15, %rcx
   movl     $8, %eax
   .align       16
.LB2_555:
##  lineno: 138
```

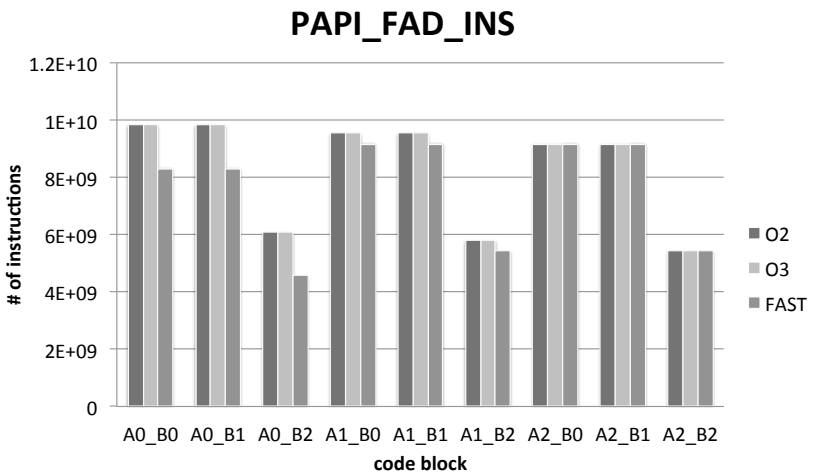
Annotated Source for `-fast`

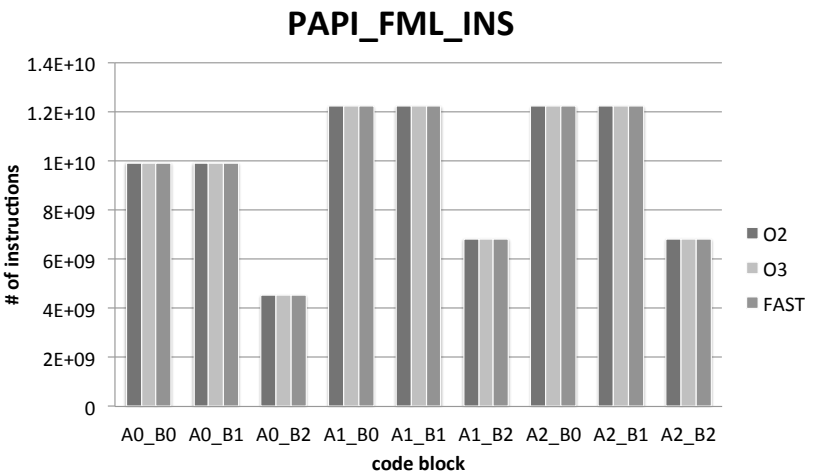
```
##       do 90 j=i+1,n-1
##       !------- Block A -------
## #if  defined(A0)
##            r2=0.0d0
##            do k=1,3
##               xx(k)=x(k,i)-x(k,j)
   movlpd   .BSS2+48(%rip),%xmm2
   movlpd   .C2_291(%rip),%xmm0
   mulsd    %xmm2,%xmm2
   addsd    %xmm1,%xmm2
   movlpd   %xmm2,344(%rsp)
   sqrtsd   %xmm2,%xmm2
   movlpd   %xmm2,448(%rsp)
   mulsd    md_globals_10_+120(%rip),%xmm2
   subsd    %xmm2,%xmm0
   .p2align  4,,1
```

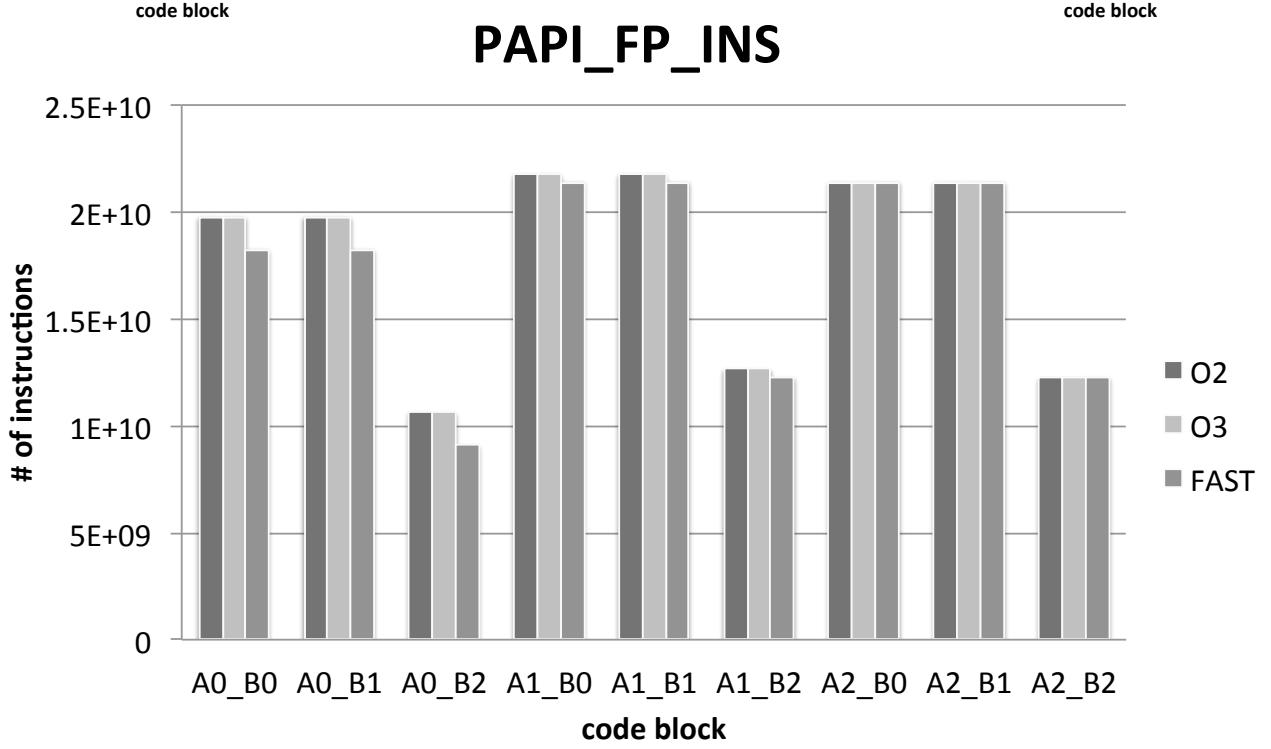
Overview

3 PAPI measurements

Floating Point Instructions







FPU idle times

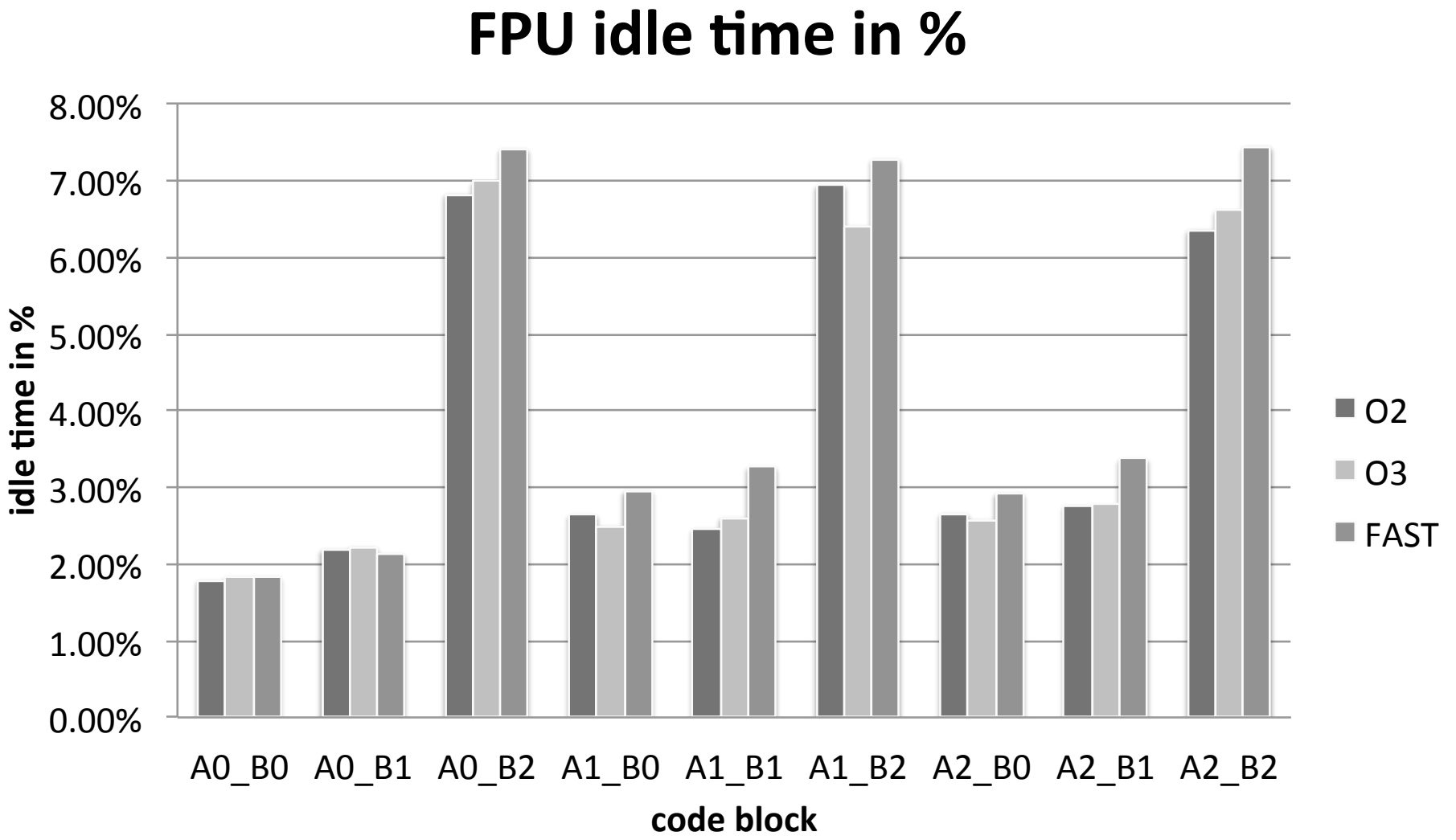
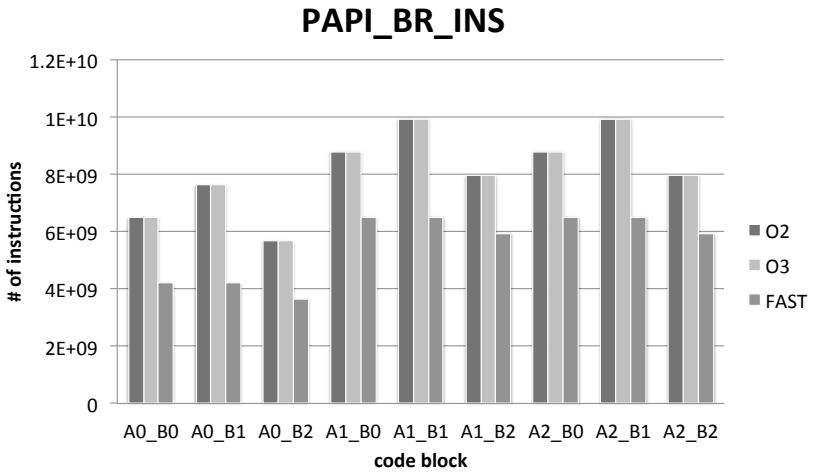
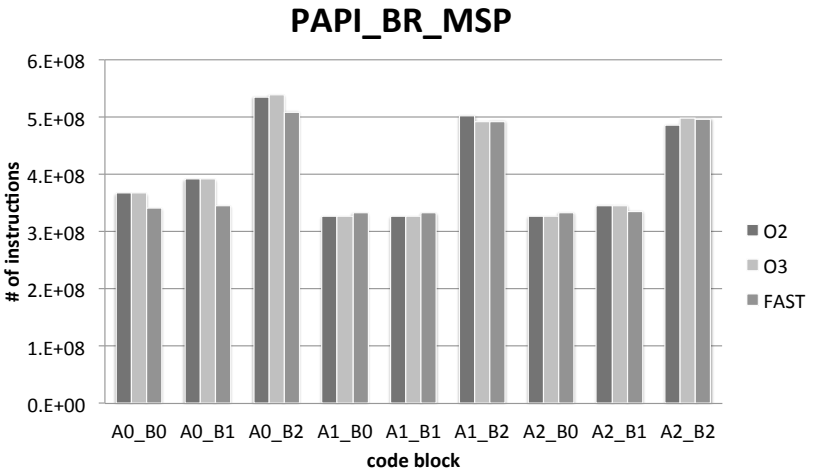
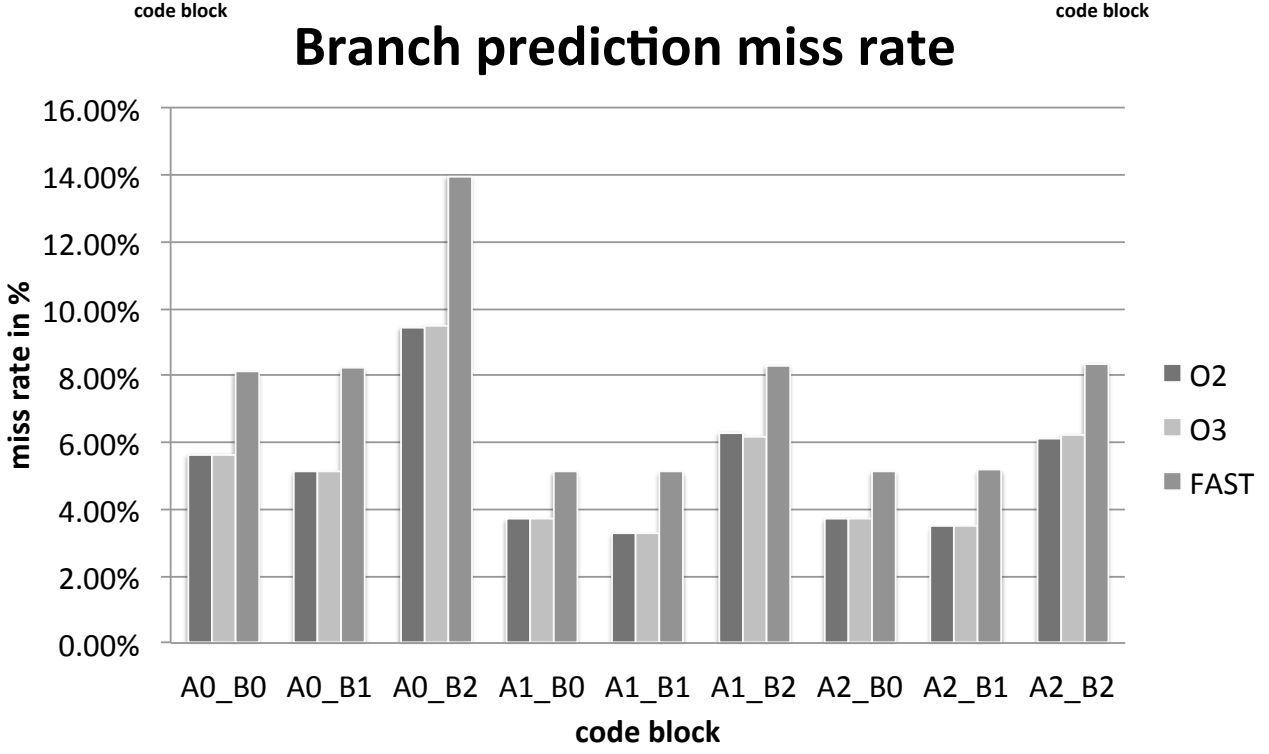


Figure : FPU idle times in percent of PAPI measured total cycles

Branch Miss Predictions







Overview

4 Source Code Analysis

Block A of the PP02 file

```
#if defined(A0)
   r2=0.0d0
   do k=1,3
      xx(k)=x(k,i)-x(k,j)
      if(xx(k).gt.+halfl(k)) xx(k)=xx(k)-xl(k)
      if(xx(k).lt.-halfl(k)) xx(k)=xx(k)+xl(k)
      r2=r2+xx(k)*xx(k)
   enddo
#elif defined(A1)
   r2=0.0d0
   do k=1,3
      xx(k)=x(k,i)-x(k,j)
      xx(k)=xx(k)-aint(xx(k)*halfli(k))*xl(k)
      r2=r2+xx(k)*xx(k)
   enddo
#elif defined(A2)
   xx(:) = x(:,i)-x(:,j)
   xx=xx-aint(xx*halfli)*xl
   r2=xx(1)*xx(1)+xx(2)*xx(2)+xx(3)*xx(3)
#else
```

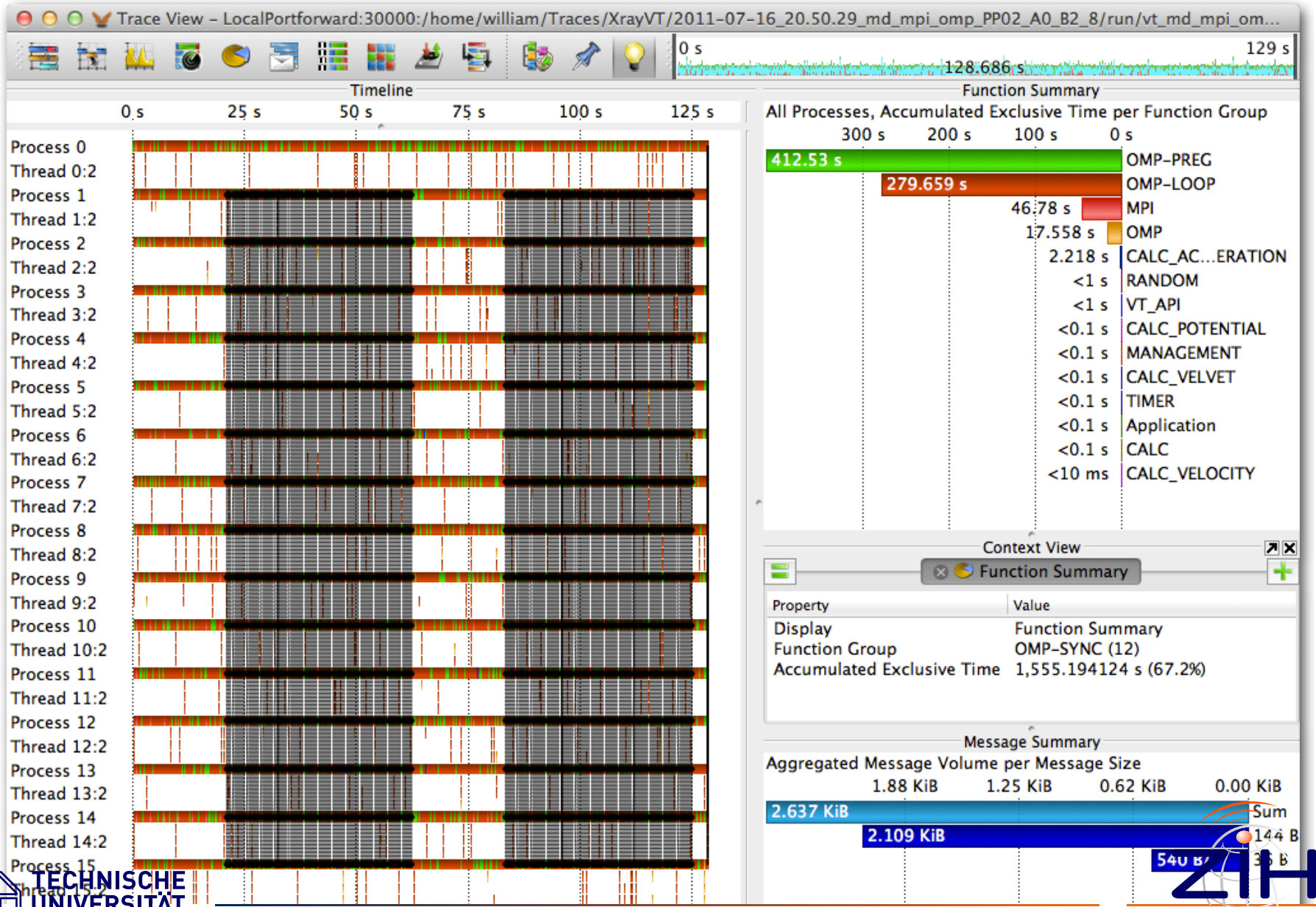
Block B of the PP02 file

```
#if defined(B0)
   r=sqrt(r2)
   fc = exp(-xmuc*r)*(1./r+xmuc)/r2
   do k=1,3
      fi(k)    = fi(k)   + zii(j)*fc*xx(k)
      fj(k,j)  = fj(k,j) - zii(i)*fc*xx(k)
   enddo
#elif defined(B1)
   r=sqrt(r2)
   fc = exp(-xmuc*r)*(1./r+xmuc)/r2
   fi(:)    = fi(:)   + zii(j)*fc*xx(:)
   fj(:,j)  = fj(:,j) - zii(i)*fc*xx(:)
#elif defined(B2)
   if(r2.le.rcutoff2) then
      r=sqrt(r2)
      fc=exp(-xmuc*r)*(1./r+xmuc)/r2
      fi(:)    = fi(:)   + zii(j)*fc*xx(:)
      fj(:,j)  = fj(:,j) - zii(i)*fc*xx(:)
   endif
#else
```

Overview

5 Source Code Optimization

OpenMP Overhead in Vampir



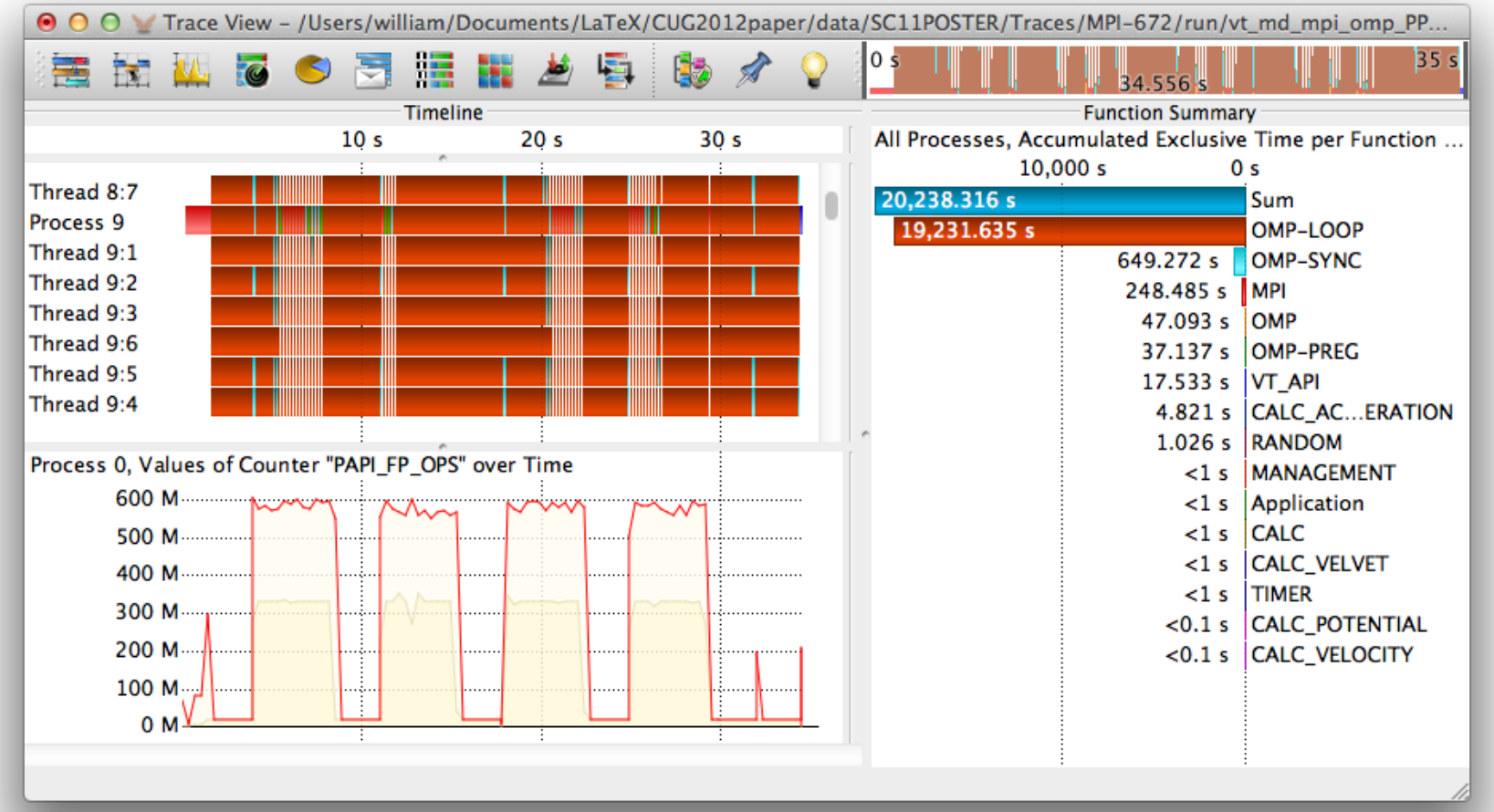
Code Changes

- `omp parallel do` for the j-loop is removed
  - `omp parallel` region around the i-loop
- j-loop is parallelized explicitly
  - Entire ij-loop section in a 3rd loop over threads
- Each iteration of this outer loop is assigned to an OpenMP thread

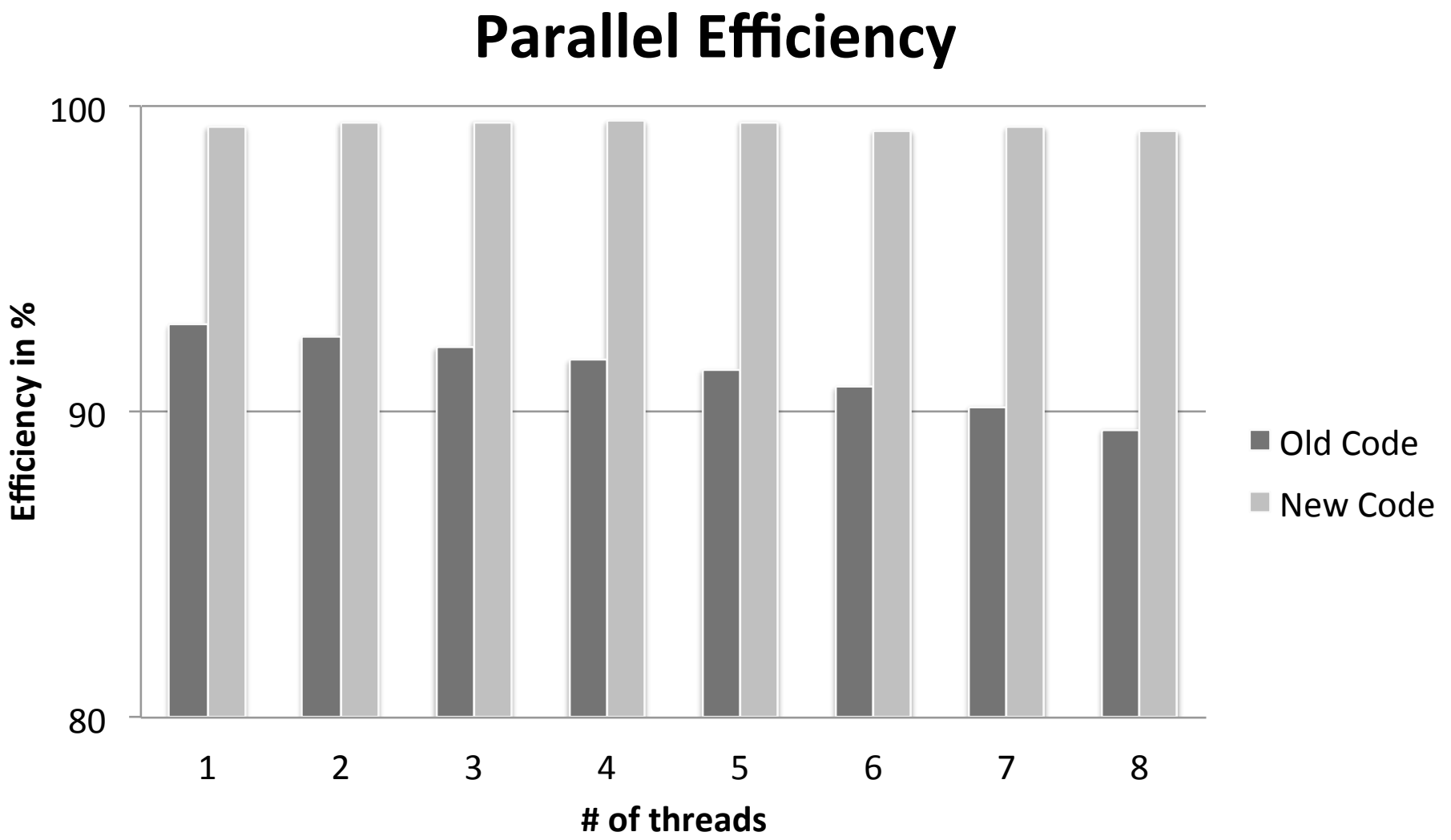
```
allocate(fj(3,0:n-1))
do 100 i=myrank,n-2,nprocs
   fi(:)=0.0d0
   !$omp parallel do private(r2,k,xx,r,fc),\
         reduction(+:fi), schedule(runtime)
   do 90 j=i+1,n-1
      !--- A-Block---
.
.
.
.
.
.
```

```
!$omp parallel private(nthrd,nj,nr,j0,
                       j1,xx,r2,r,fc,fi)
nthrd = omp_get_num_threads()
allocate(fi(3,0:n-1))
!$omp do schedule(static,1)
do 110 ithrd=0,nthrd-1
do 100 i=myrank,n-2,nprocs
   fi(:,i)=0.0d0
   nj=(n-i-1)/nthrd
   nr=mod(n-i-1,nthrd)
   j0=(i+1)+ithrd*nj+min(ithrd,nr)
   j1=(i+1)+(ithrd+1)*nj+min(ithrd+1,nr)-1
   do 90 j=j0,j1
```

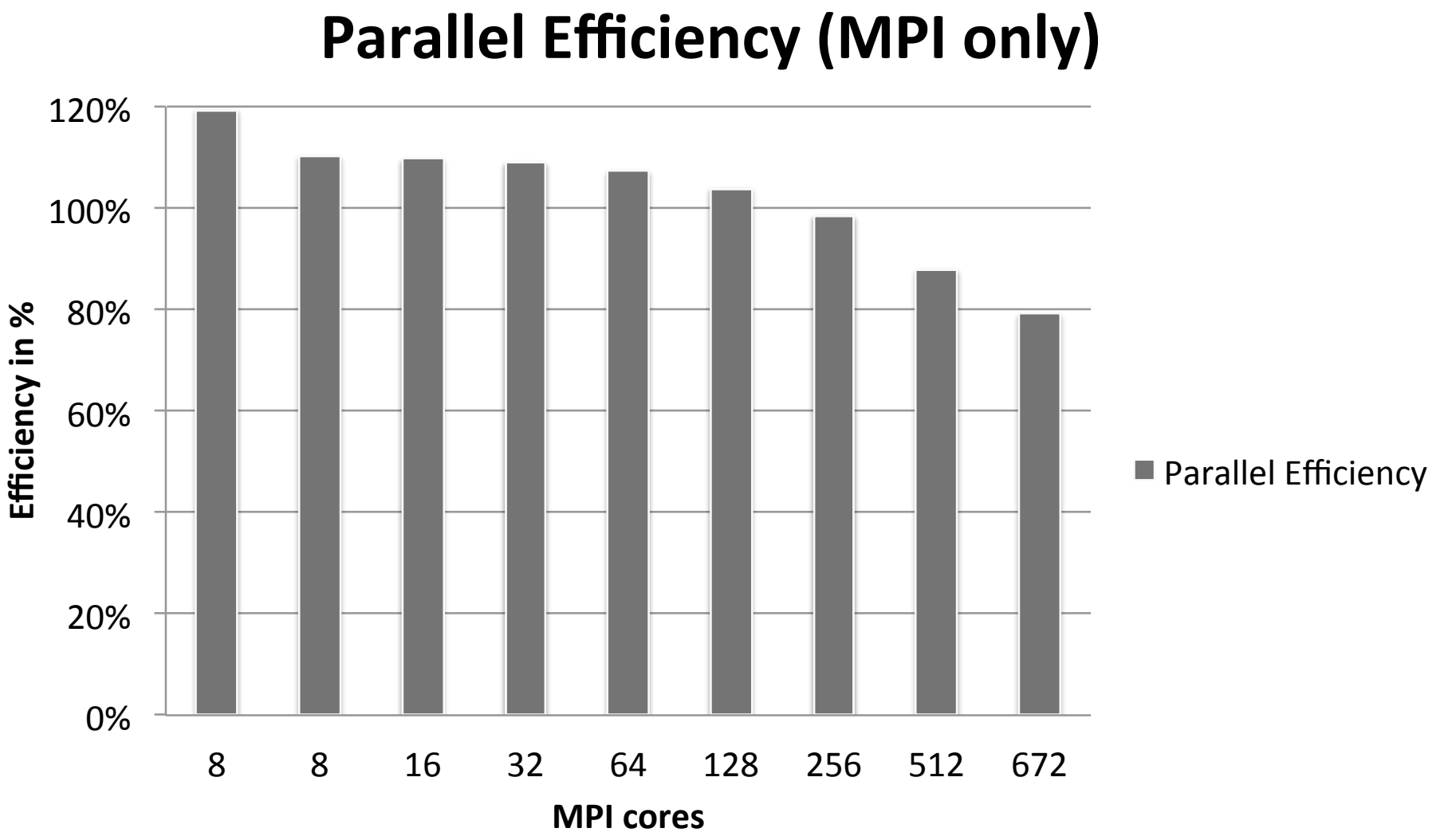
New MPI/OpenMP Version in Vampir



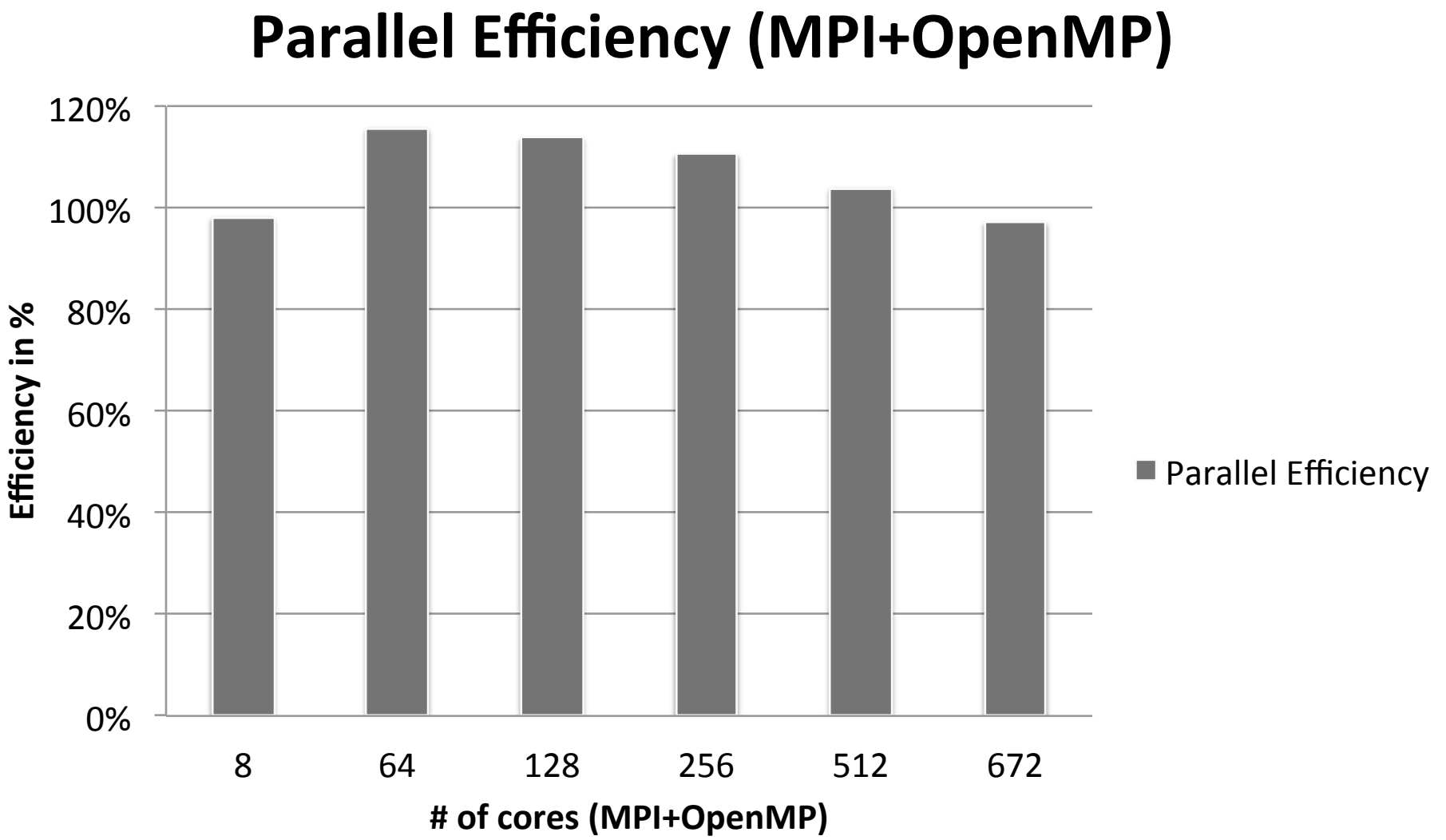
Parallel Efficiency of the OpenMP Application



Parallel Efficiency of the MPI Version



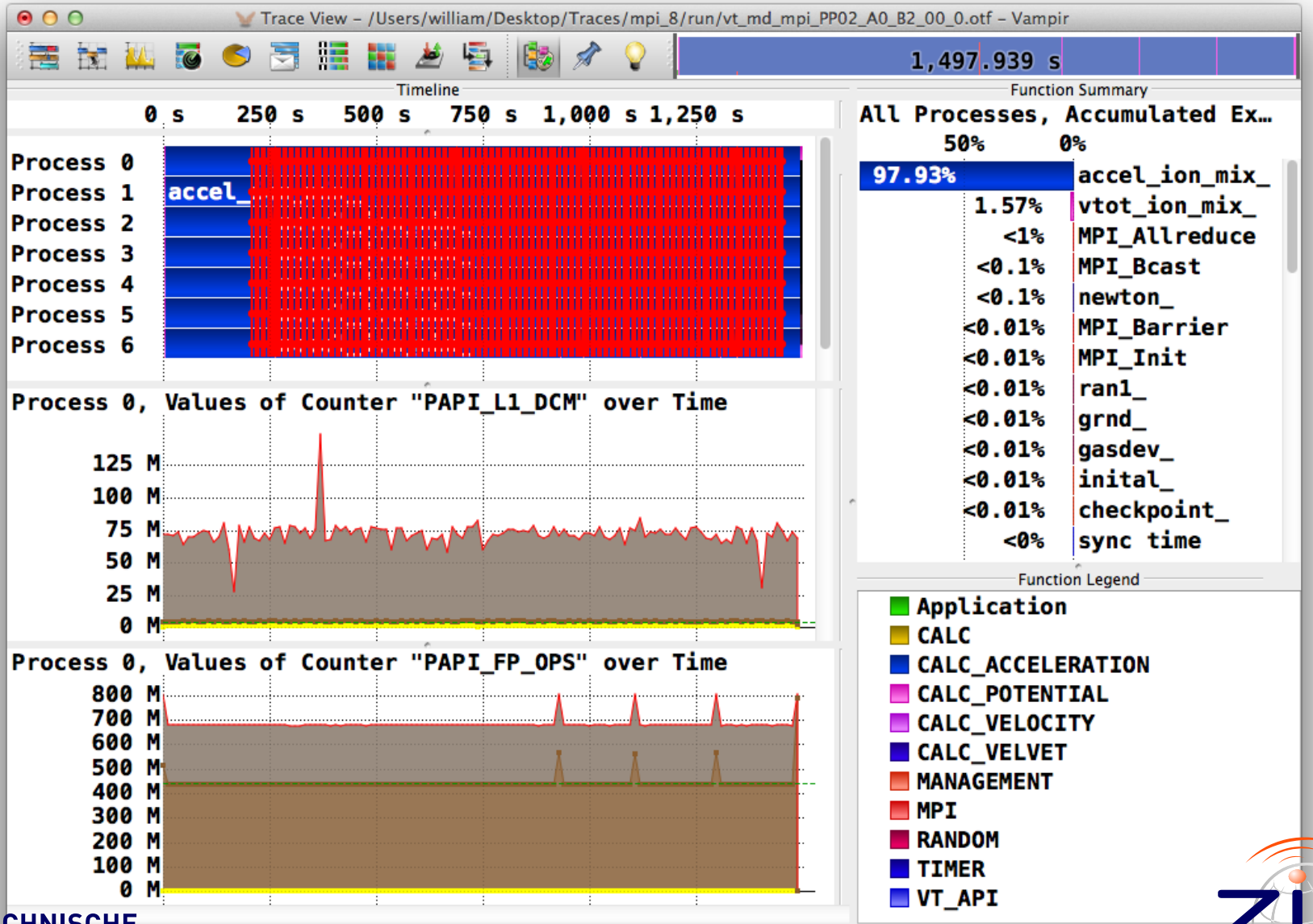
Parallel Efficiency of the Hybrid Version



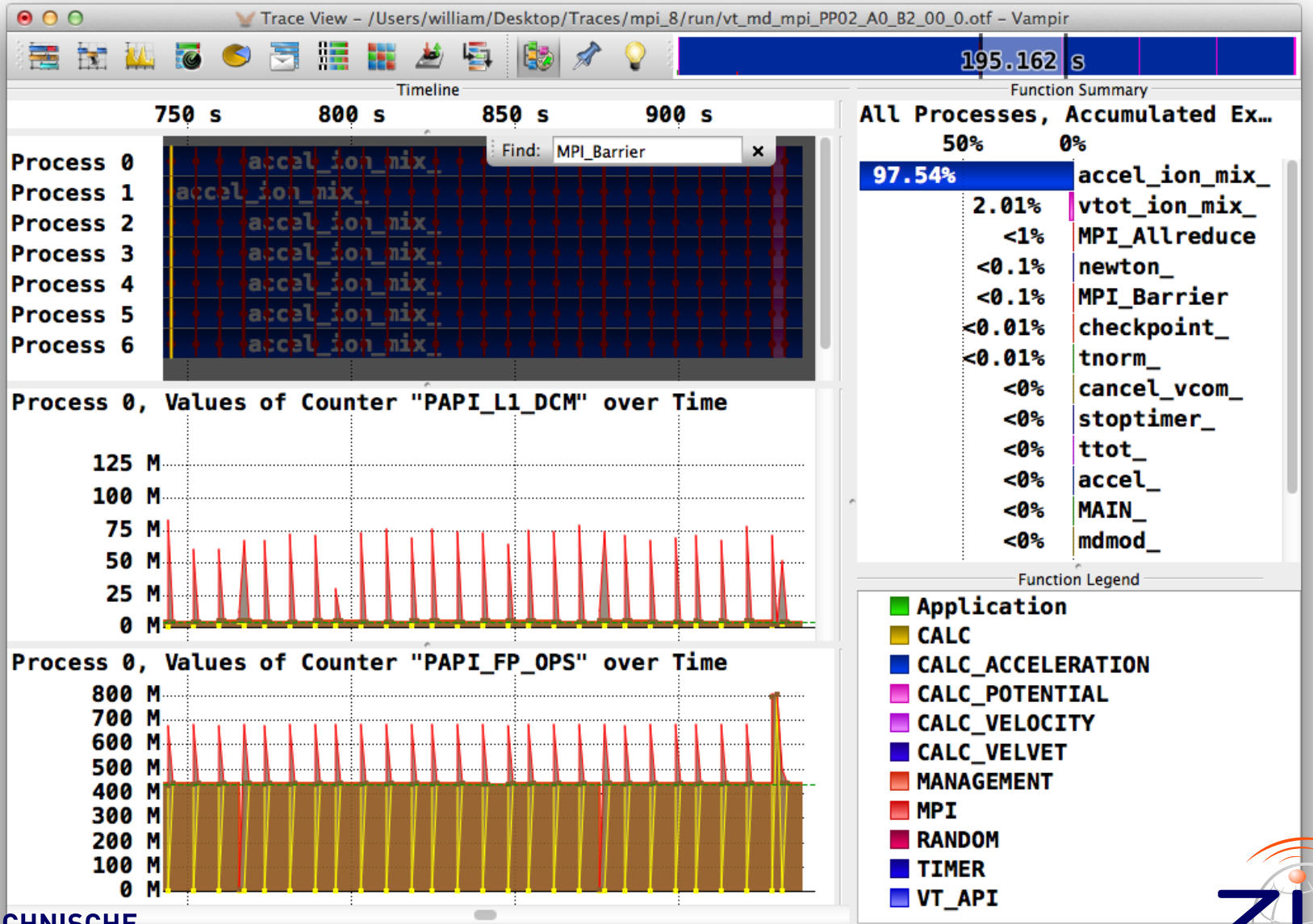
Overview

6 Tracing and Visualization

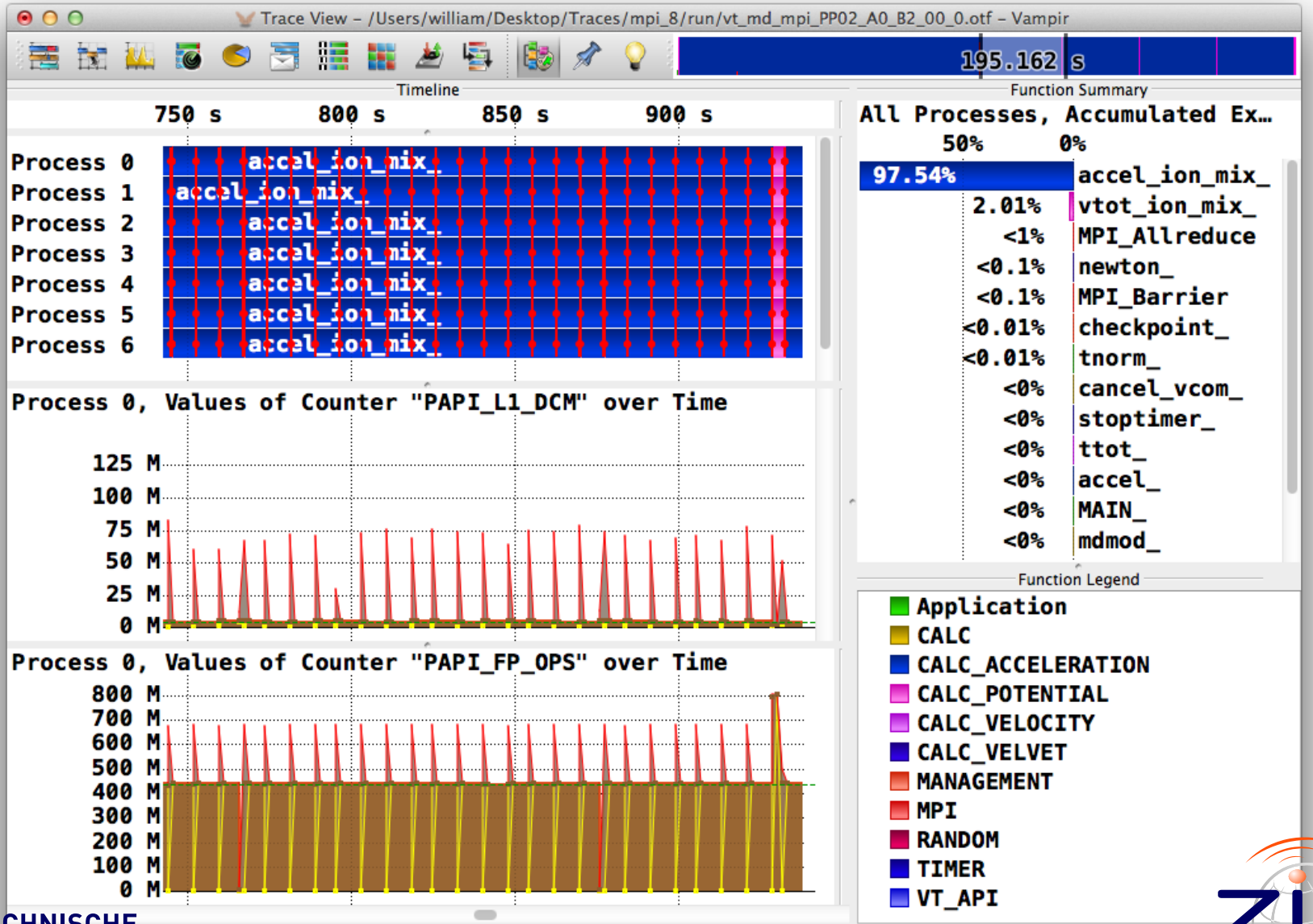
8 core inter-node run of the MPI-only version of MD



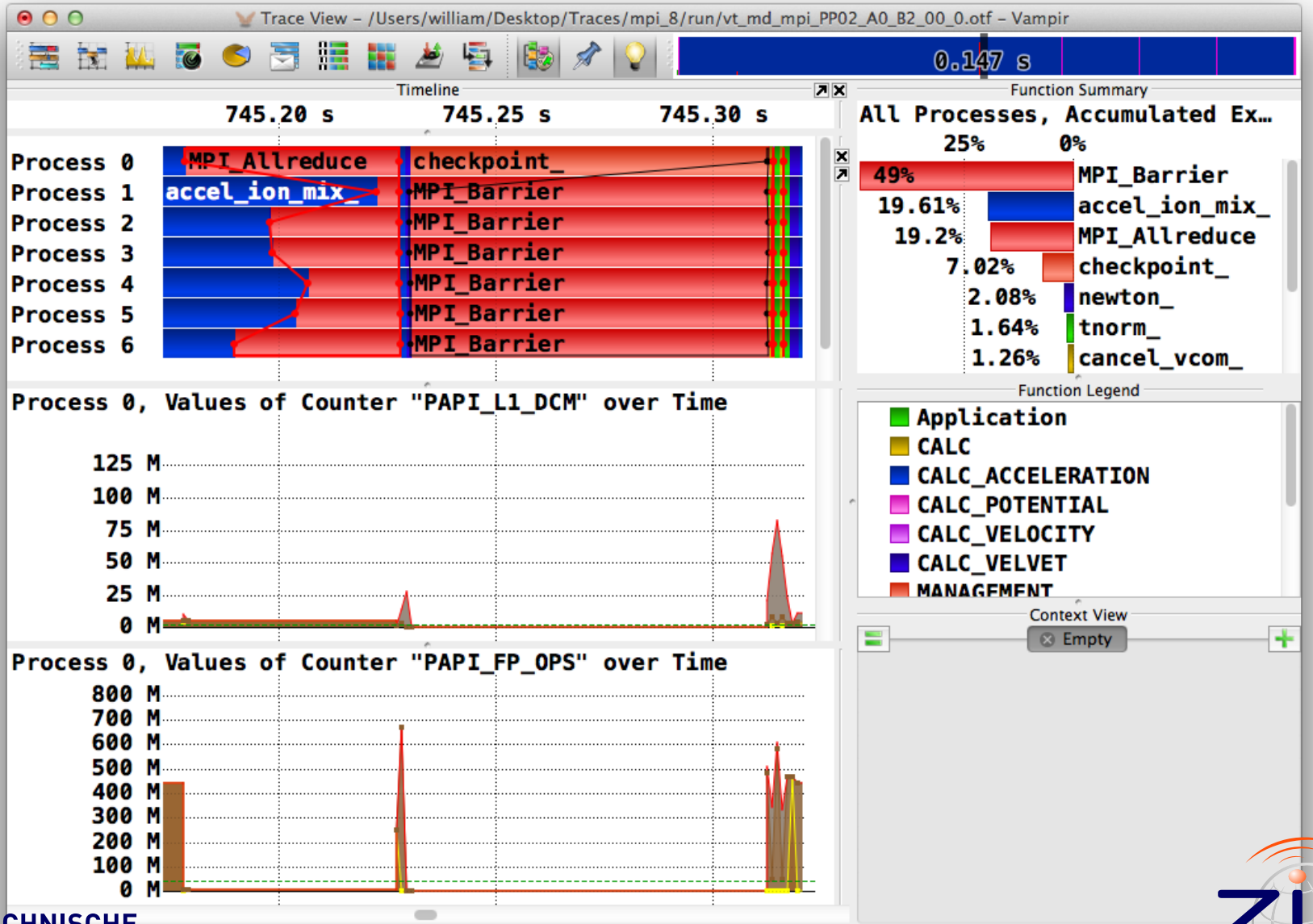
8 core inter-node run of the MPI-only version of MD



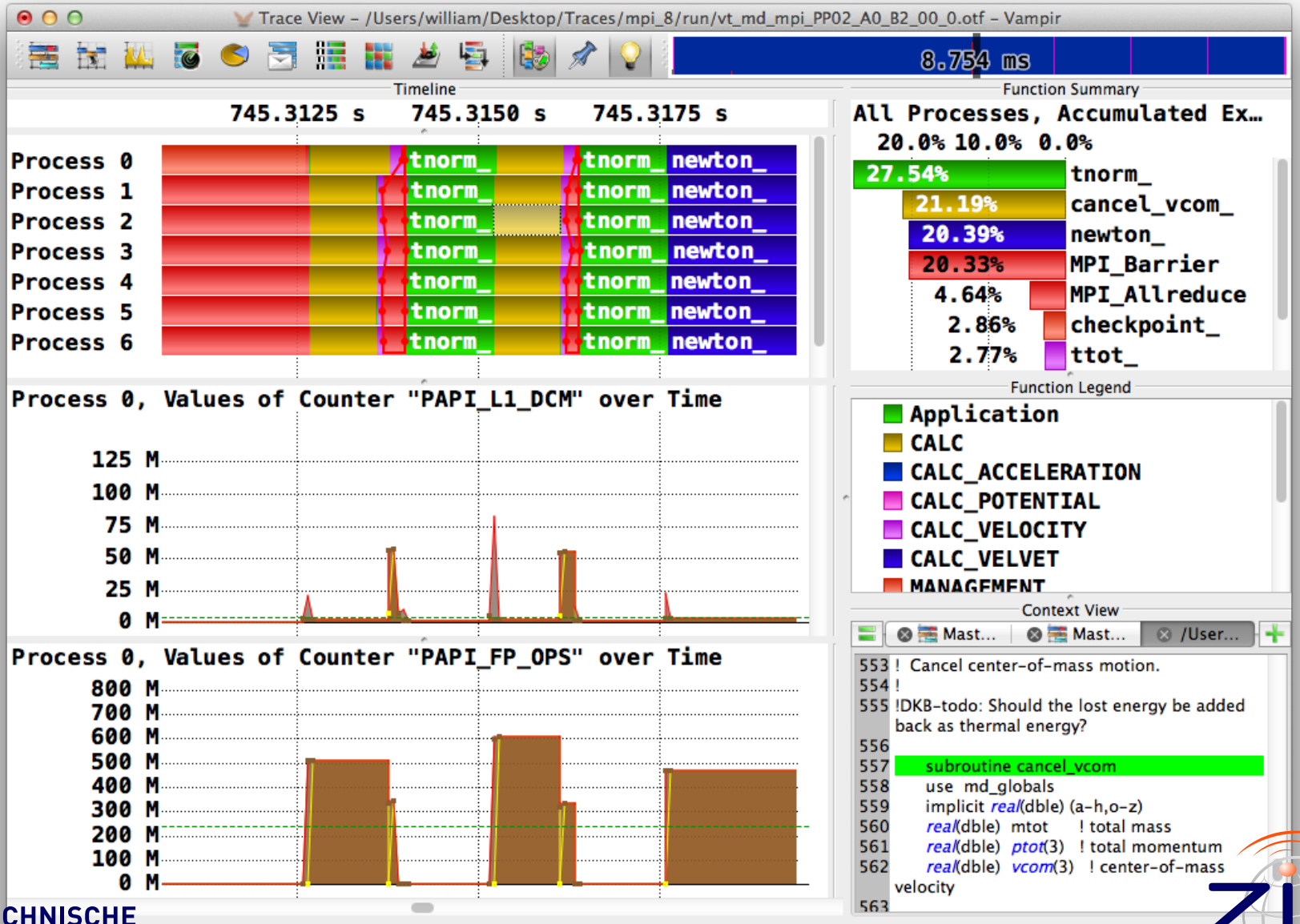
8 core inter-node run of the MPI-only version of MD



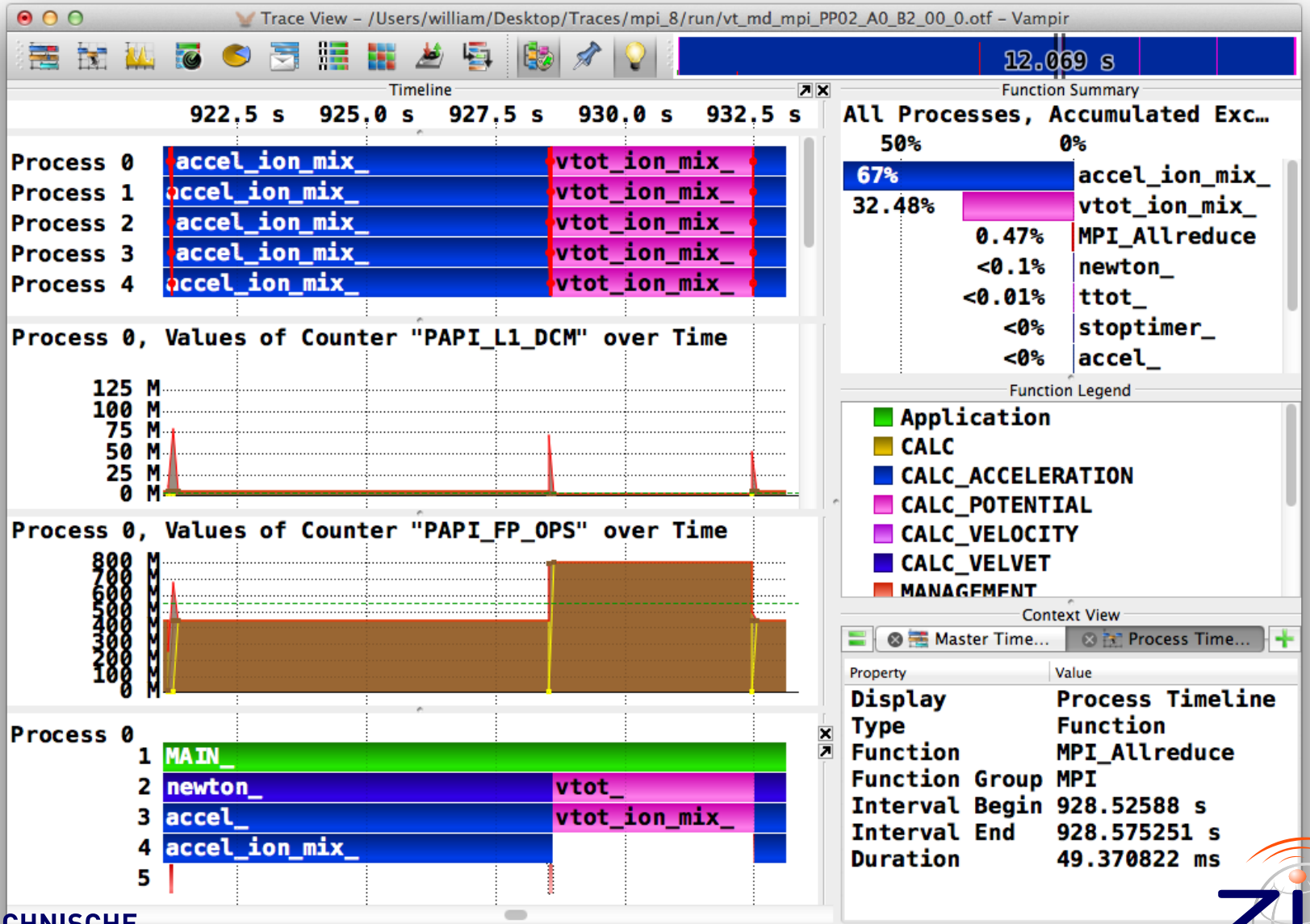
8 core inter-node run of the MPI-only version of MD



8 core inter-node run of the MPI-only version of MD



8 core inter-node run of the MPI-only version of MD



Impact of `MPI_Allreduce` on the scaling

Table : MPI-only, PP02 A0 B2 run, all processes, accumulated exclusive time per function

function name	672 cores time in s	%	8 cores time in s	%
accel_ion_mix	13293.4	72.7	11735.0	97.9
MPI_Allreduce	1885.8	10.3	44.1	0.4
sync	939.4	5.1	0.1	0
newton	937.2	5.1	4.9	0.0
MPI_Bcast	807.1	4.4	8.2	0.1
vtot_ion_mix	189.6	1.0	188.2	1.6
MPI_Init	112.8	0.6	0.1	0
MPI_Barrier	107.5	0.6	1.2	0

Overview

7 Conclusion

Conclusion

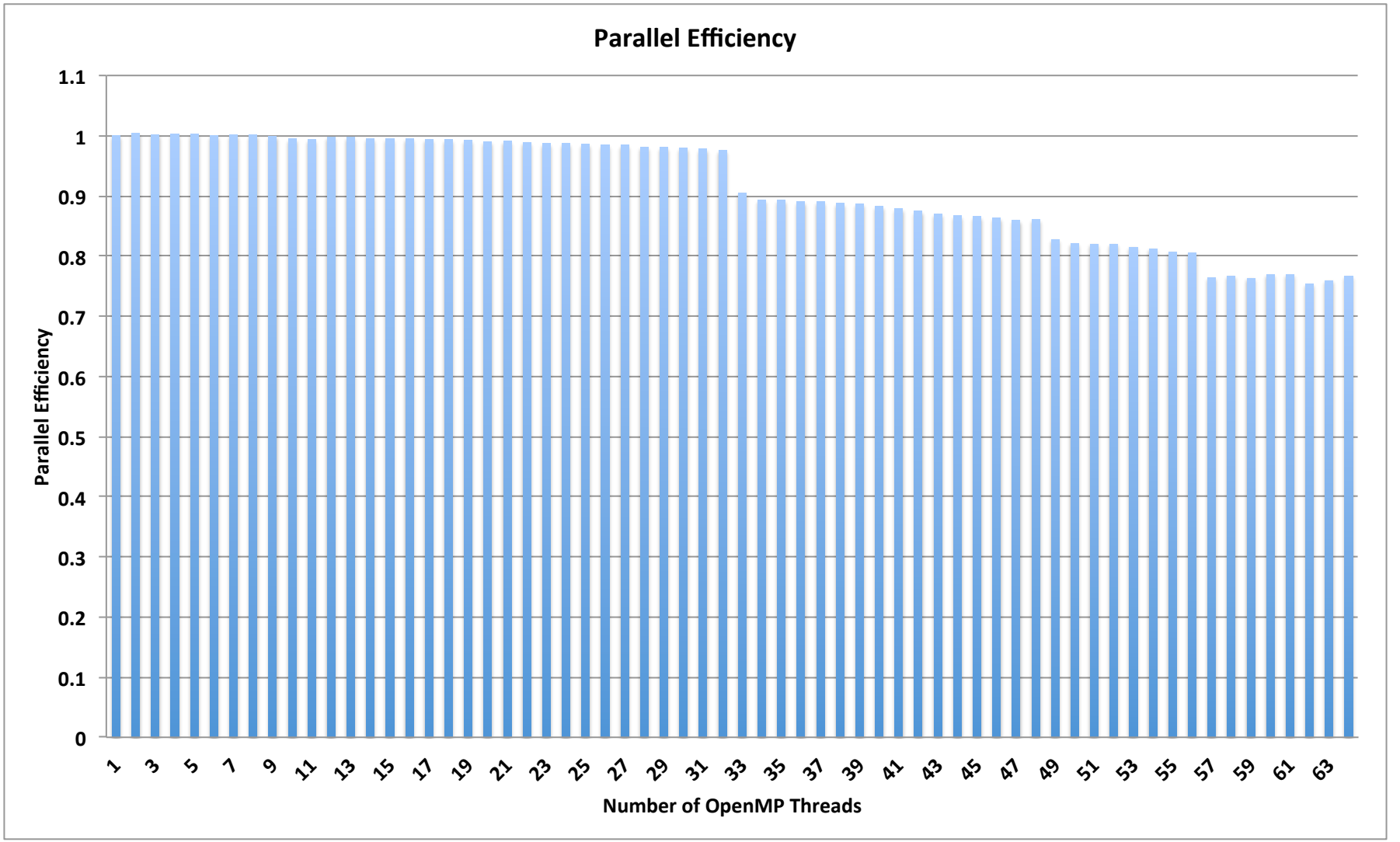
- The analysis of MD found the best serial version
- Using a cut-off sphere reduces interactions to 19%
- Runtime decrease although the performance counter indicate otherwise
- Moved the OpenMP parallelization from the inner to the outer loop
- Improved the parallel efficiency
- Dual socket, dual chip 16-threads AMD Interlagos system shows a 97.6% parallel efficiency up to 32 cores

The changes to the source code will be included in future versions of the Spec OpenMP benchmark.

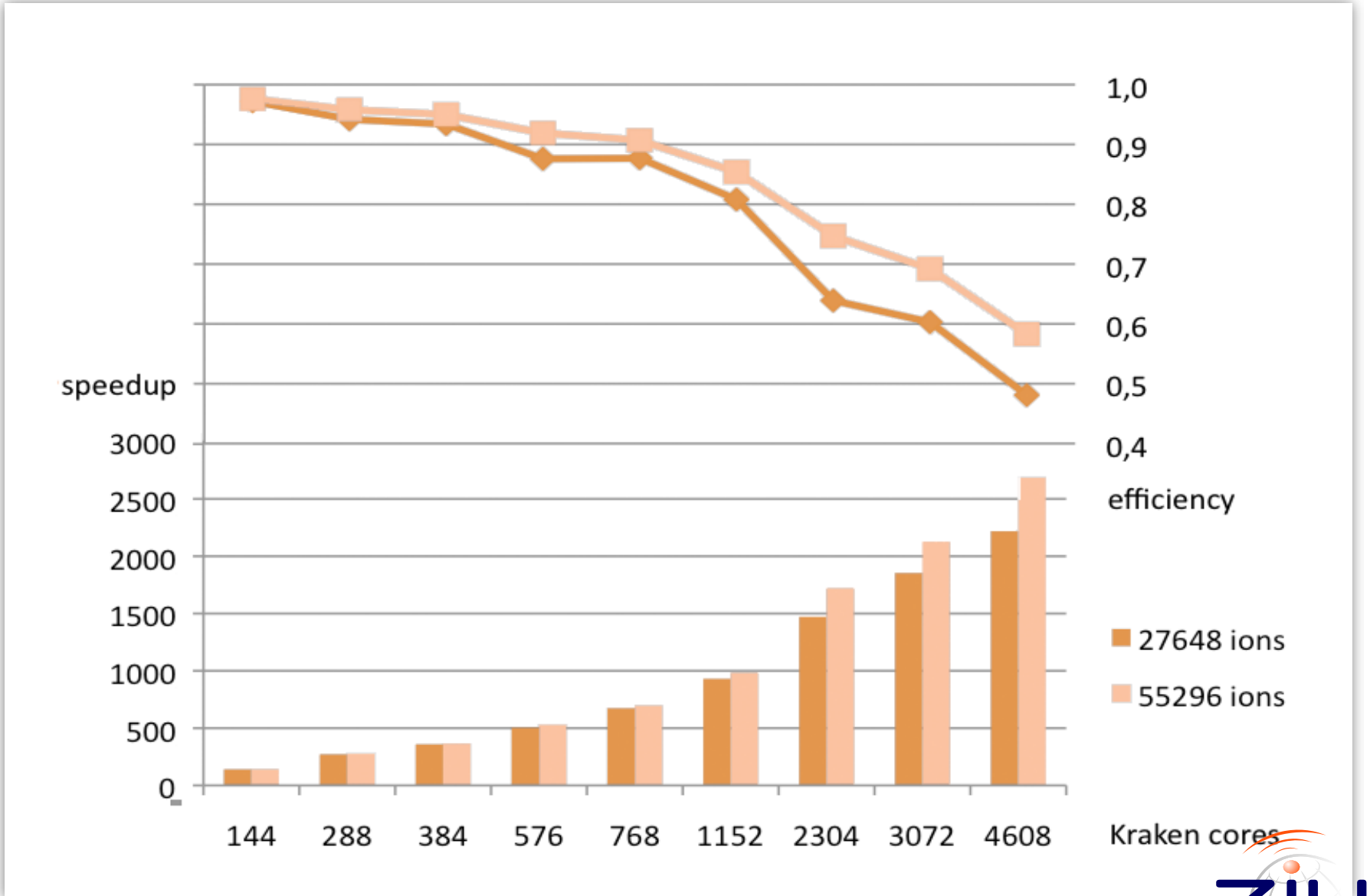
Questions?

Thank you for your attention.

AMD Interlagos OpenMP Efficiency



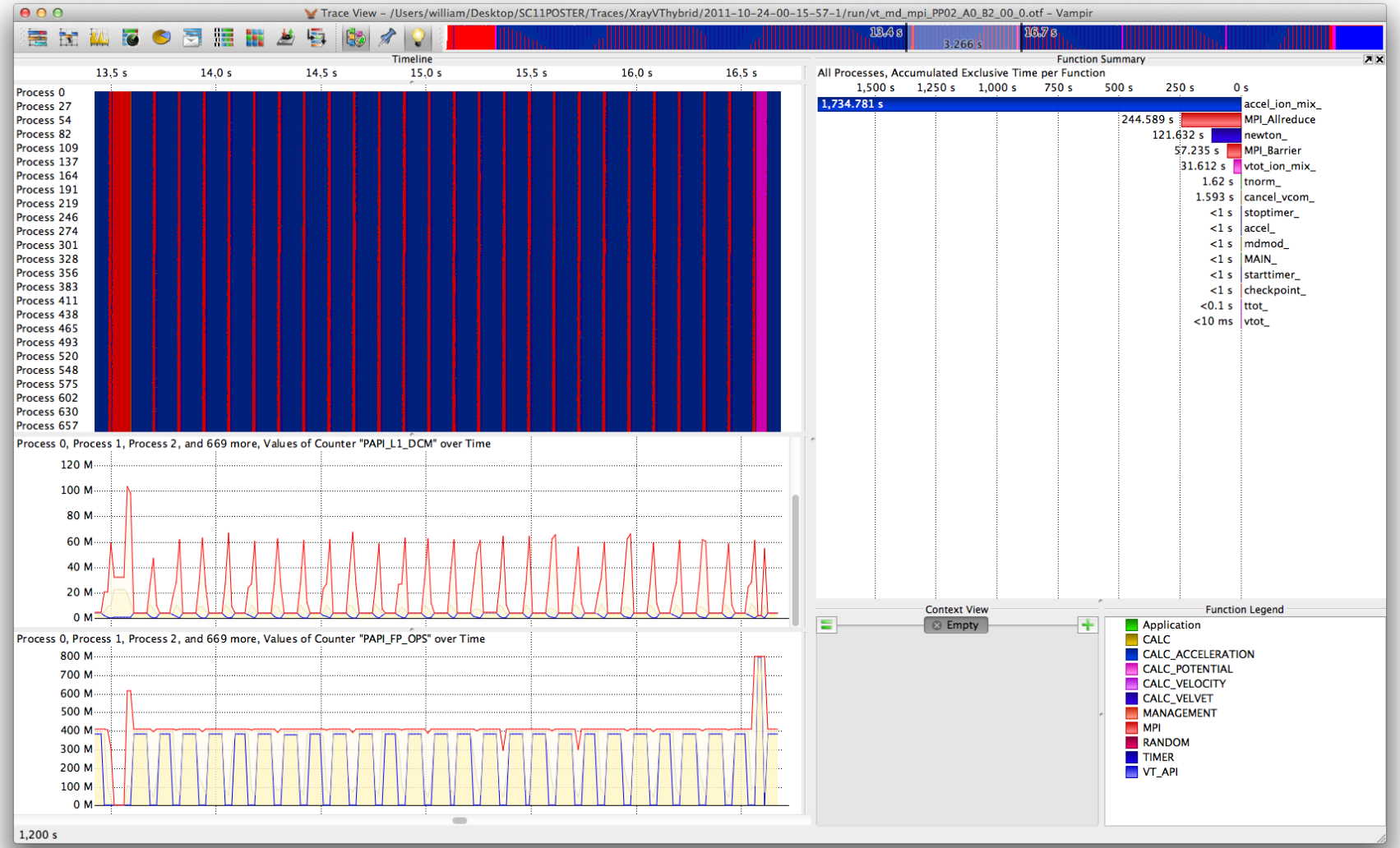
Kraken - Old Version



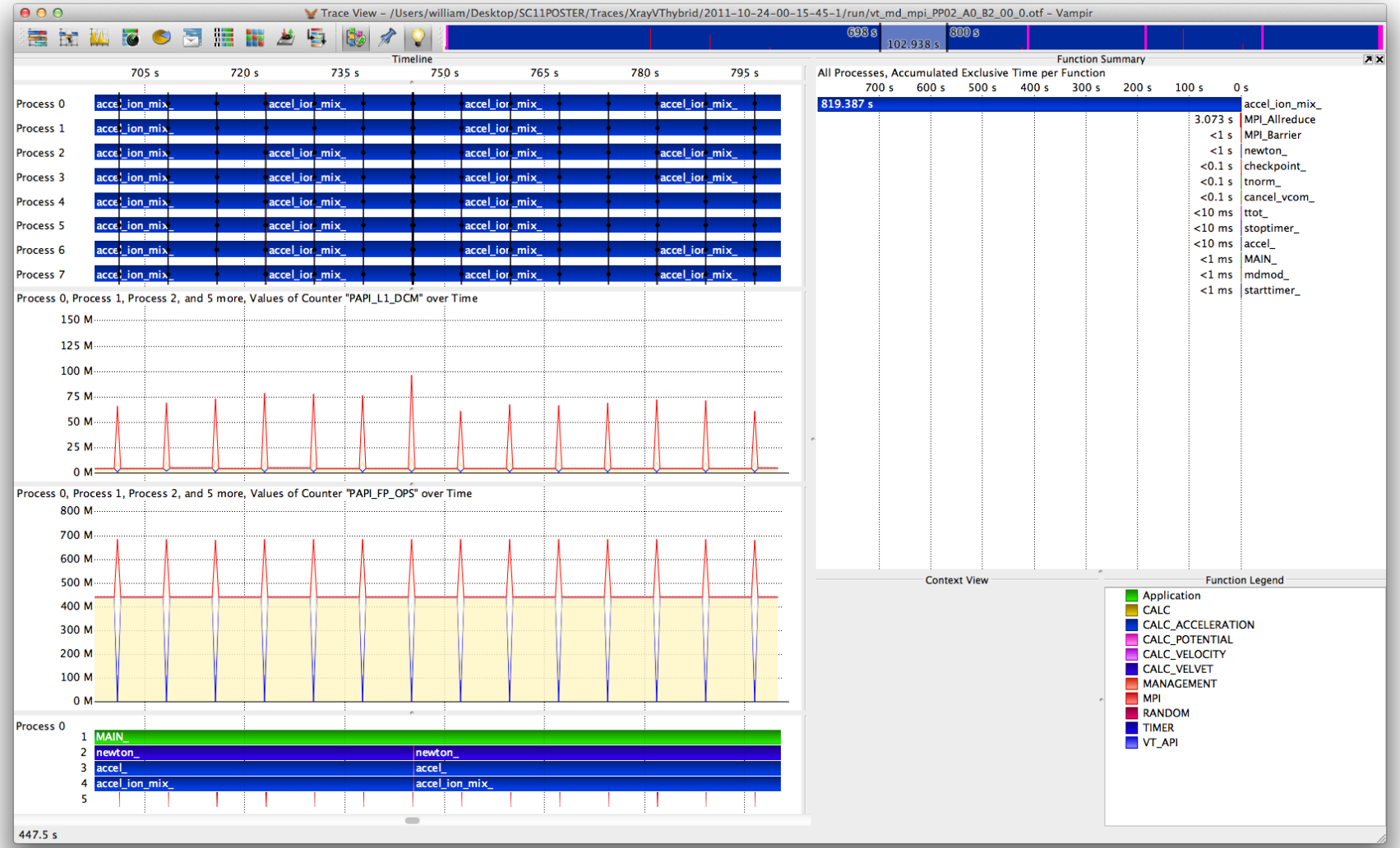
Introduction

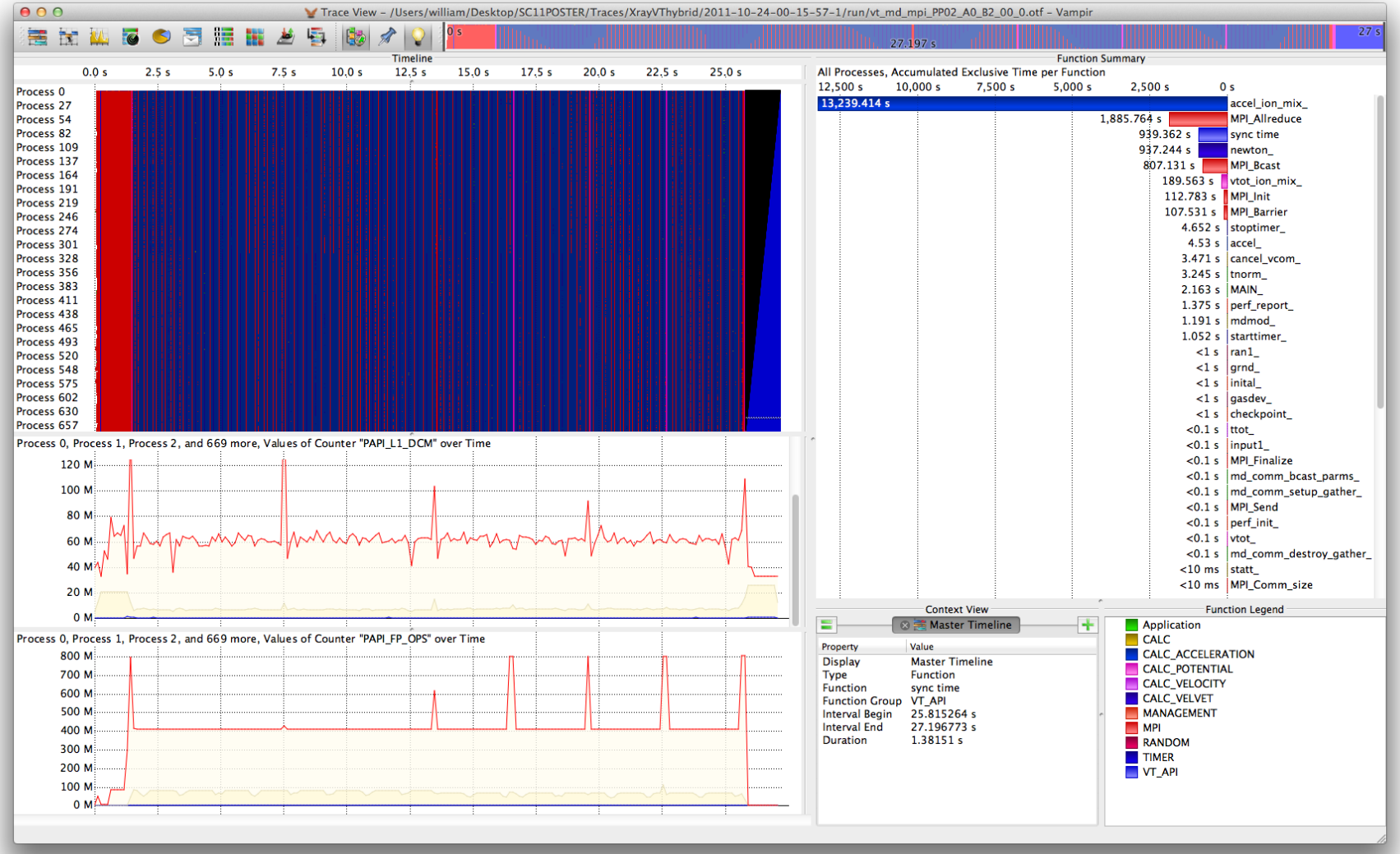
- Code has been extended and rewritten multiple times
- Several different implementations of the same semantics
  - Loop reimplemented using the Fortran array syntax
  - Cut-off sphere to restrict interactions to a subset of relatively nearby nucleons/ion
- User must specify which variation in the force calculation routines to use when building the code
  - nucleon
  - pure-ion
  - ion-mixture

672 core run of the MPI-only version of MD



8 core inter-node run of the MPI-only version of MD




Trace View – /Users/william/Desktop/SC11POSTER/Traces/XrayVThybrid/2011-10-24-00-15-57-1/run/vt_md_mpi_PP02_A0_B2_00_0.otf – Vampir
Timeline
Function Summary
All Processes, Accumulated Exclusive Time per Function
Process 0, Process 1, Process 2, and 669 more, Values of Counter "PAPI_L1_DCM" over Time
Process 0, Process 1, Process 2, and 669 more, Values of Counter "PAPI_FP_OPS" over Time
Context View
Master Timeline
Function Legend
Application
CALC
CALC_ACCELERATION
CALC_POTENTIAL
CALC_VELOCITY
CALC_VELVET
MANAGEMENT
MPI
RANDOM
TIMER
VT_API