CRAY
THE SUPERCOMPUTER COMPANY

The Cray Compiler Environment: Introduction and Intial Results

Nathan Wichmann
wichmann@cray.com

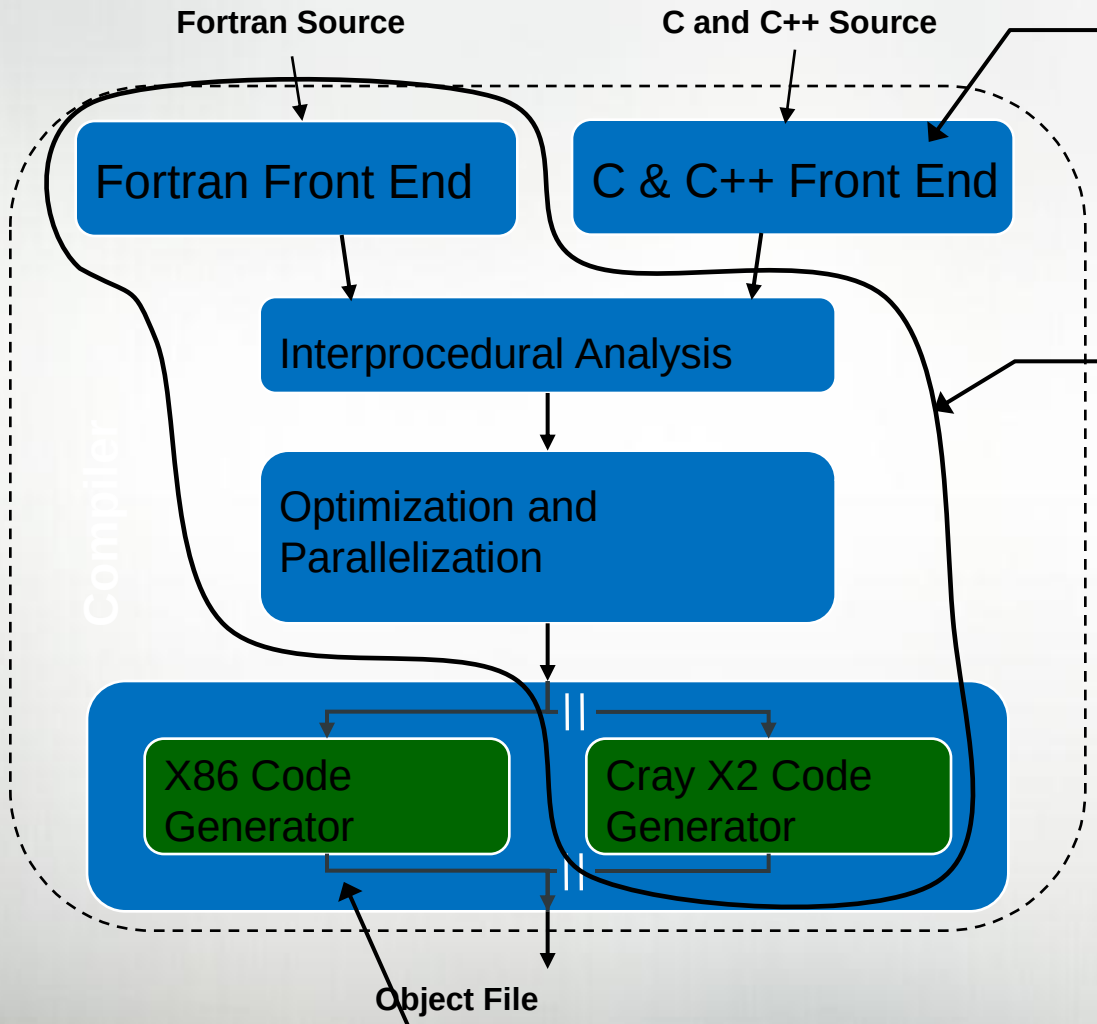
Outline

- Introduction to the Cray compiler
- Example
  - GTC
  - Overflow
  - PARQUET

Cray Compiler Environment (CCE): Brief History of Time

- Cray has a long tradition of high performance compilers
  - Vectorization
  - Parallelization
  - Code transformation
  - More…
- Began internal investigation leveraging an open source compiler called LLVM
- Initial results and progress better than expected
- Decided to move forward with Cray X86 compiler
- 7.0 released in December 2008
- 7.1 will be released Q2 2009

Technology Sources



C and C++ Front End supplied by Edison Design Group, with Cray-developed code for extensions and interface support

Cray Inc. Compiler Technology

X86 Code Generation from Open Source **LLVM**, with additional Cray-developed optimizations and interface support

Cray Opteron Compiler: How to use it

- Make sure it is available
  - module avail PrgEnv-cray
- To access the Cray compiler
  - module load PrgEnv-cray
- To target the Barcelona chip
  - module load xtpe-quadcore
- Once you have loaded the module “cc” and “ftn” are the Cray compilers
  - Recommend just using default options
  - Use –rm (fortran) and –hlist=m (C) to find out what happened
- man crayftn

Cray Opteron Compiler: Current Capabilities

- Excellent Vectorization
  - Vectorize more loops than other compilers
- OpenMP
  - 2.0 standard
  - Nesting
- PGAS: Functional UPC and CAF available today.
- Excellent Cache optimizations
  - Automatic Blocking
  - Automatic Management of what stays in cache
- Prefetching, Interchange, Fusion, and much more…

Cray Opteron Compiler: Future Capabilities

- C++ Support
- Automatic Parallelization
  - Modernized version of Cray X1 streaming capability
  - Interacts with OMP directives
- OpenMP 3.0
- Optimized PGAS
  - Will require Gemini network to really go fast
- Improved Vectorization
- Improve Cache optimizations

Case Study: The Gyrokinetic Toroidal Code (GTC)

- Plasma Fusion Simulation
- 3D Particle-in-cell code (PIC) in toroidal geometry
- Developed by Prof. Zhihong Lin (now at UC Irvine)
- Code has several different characteristics
  - Stride-1 copies
  - Strided memory operations
  - Computationally intensive
  - Gather/Scatter
  - Sorting and Packing
- Main routine is known as the “pusher”

Case Study: The Gyrokinetic Toroidal Code (GTC)

- Main Pusher kernel consists of 2 main loop nests
- First loop nest contains groups of 4 statements which include significant indirect addressing

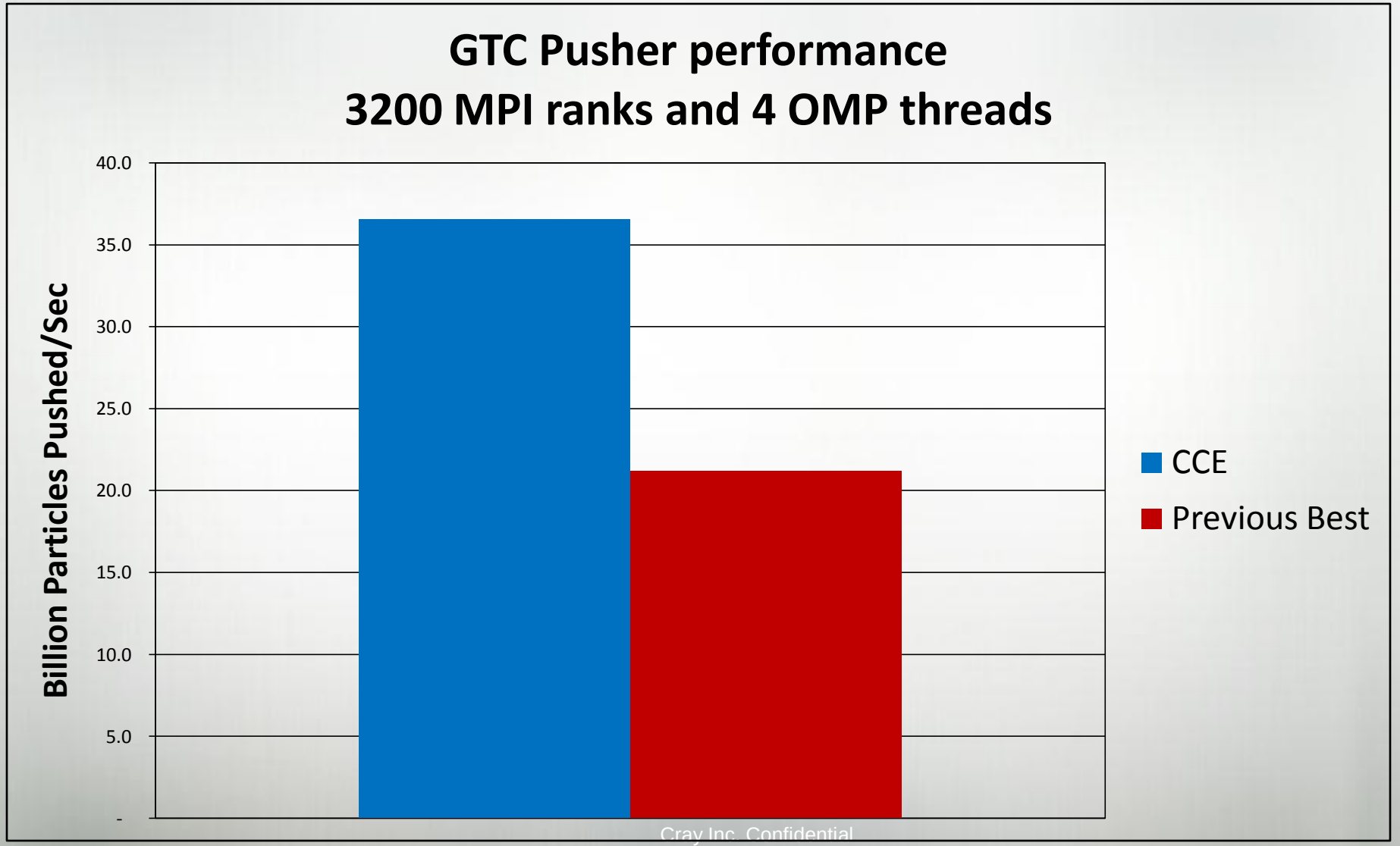
```
e1=e1+wp0*wt00*(wz0*gradphi(1,0,ij)+wz1*gradphi(1,1,ij))
e2=e2+wp0*wt00*(wz0*gradphi(2,0,ij)+wz1*gradphi(2,1,ij))
e3=e3+wp0*wt00*(wz0*gradphi(3,0,ij)+wz1*gradphi(3,1,ij))
e4=e4+wp0*wt00*(wz0*phit(0,ij)+wz1*phit(1,ij))
```

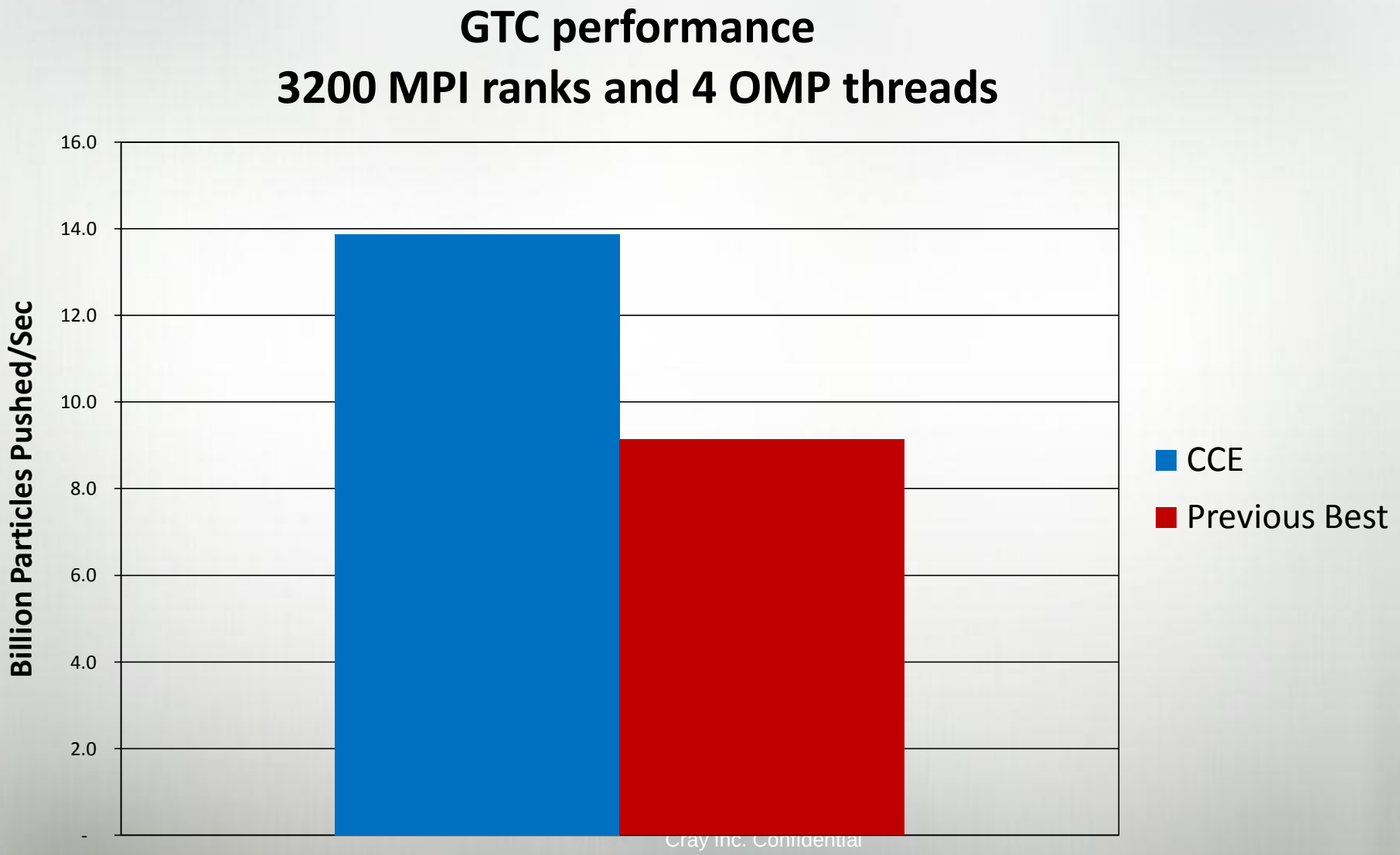
  - Turn 4 statements into 1 vector shortloop

```
ev(1:4)=ev(1:4)+wp0*wt00*(wz0*tempphi(1:4,0,ij)+wz1*tempphi(1:4,1,ij))
```

- Second loop is large, computationally intensive, but contains strided loads and computed gather
  - CCE automatically vectorizes loop

Case Study: GTC

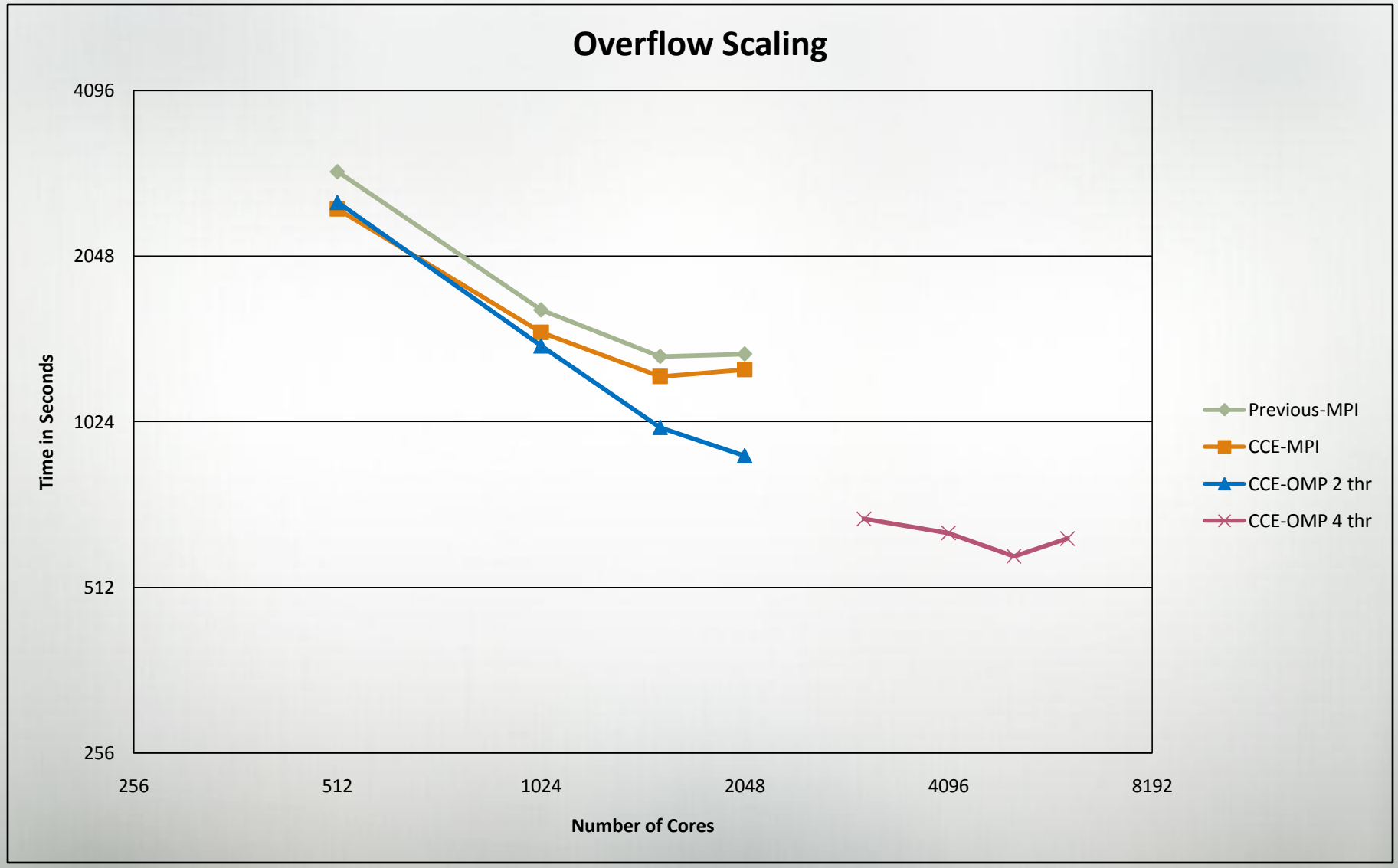




Case Study: Overflow

- Overflow is a NASA developed Navier-Stokes flow solver for unstructured grids
- Subroutines consist of two or three simply-nested loops
- Inner loops tend to be highly vectorized and have 20-50 Fortran statements
- MPI is used for parallel processing
  - Solver automatically splits grid blocks for load balancing
  - Scaling is limited due to load balancing at > 1024
- Code is threaded at a high-level via OpenMP

Overflow Scaling using only MPI vs MPI & OMP



Case Study: PARQUET

- Materials Science code
- Scales to 1000s of MPI ranks before it runs out of parallelism
- Want to use shared memory parallelism across entire node

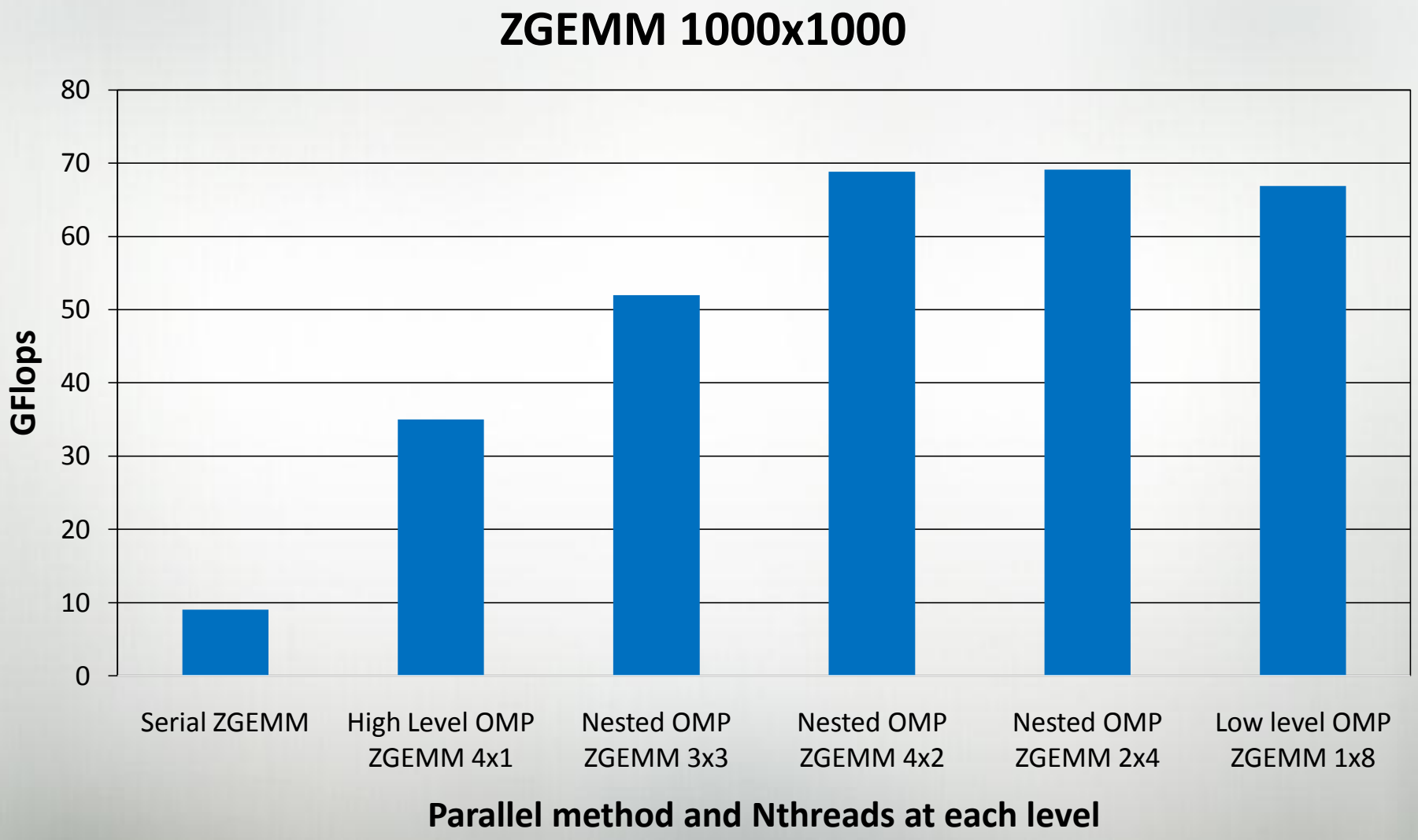
- Main kernel consists of 4 independent zgemms
- Want to use multi-level OMP to scale across the node

Case Study: PARQUET

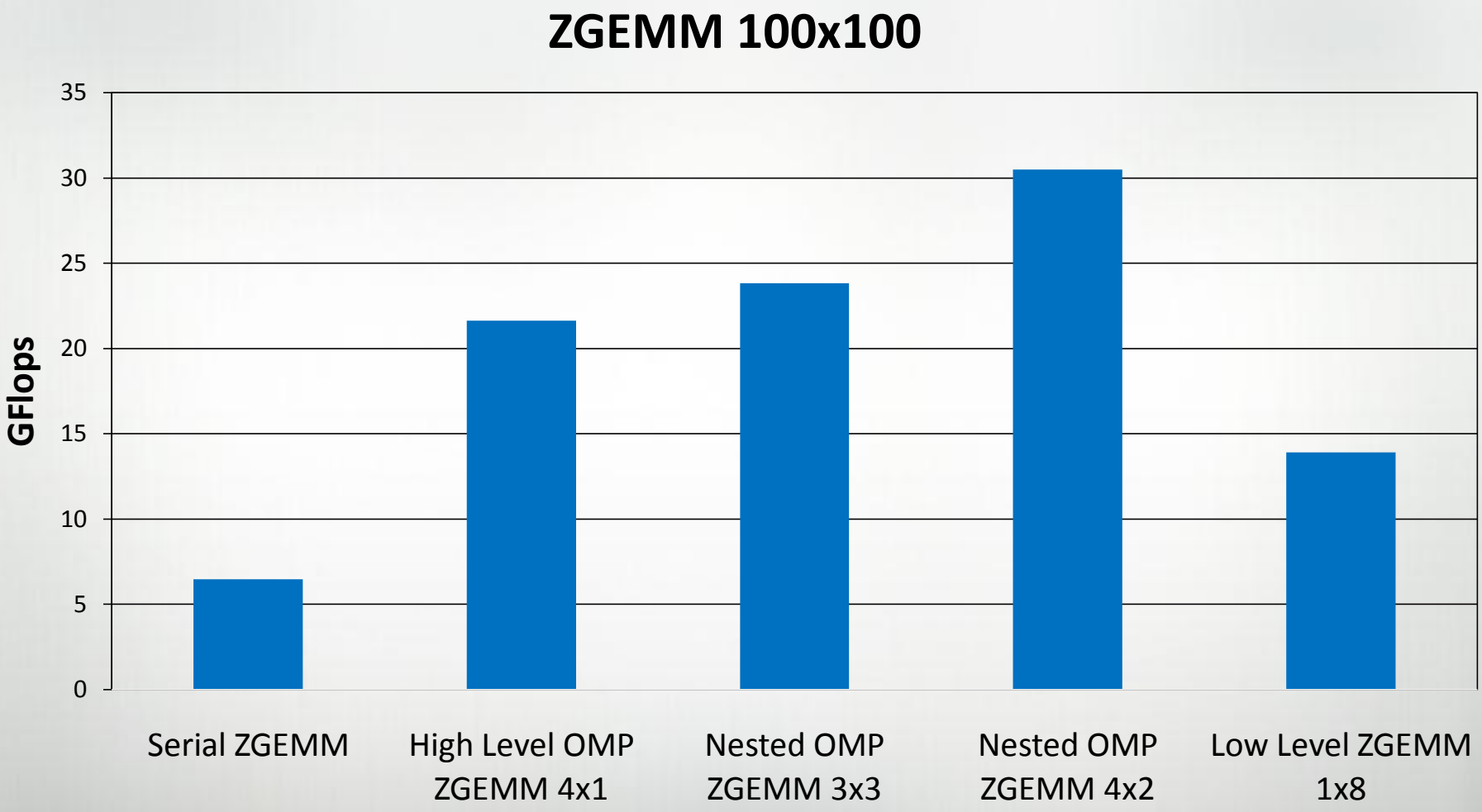
```
!$omp parallel do …
do i=1,4
    call complex_matmul(…)
enddo

Subroutine complex_matmul(…)
!$omp      parallel do private(j,jend,jsize)! num_threads(p2)
    do j=1,n,nb
      jend = min(n, j+nb-1)
      jsize = jend - j + 1
      call zgemm( transA,transB, m,jsize,k,                    &
            alpha,A,ldA,B(j,1),ldb, beta,C(1,j),ldC)
    enddo
```

Case Study: PARQUET



Case Study: PARQUET



Conclusions

- The Cray Compiling Environment is a new, different, and interesting compiler with several unique capabilities
- Several codes are already taking advantage of CCE
- Development is ongoing
- Consider trying CCE if you think you could take advantage of its capabilities


CRAY
THE SUPERCOMPUTER COMPANY