Why PaRSEC is the right runtime for exascale computing

Anthony Danalis
Innovative Computing Laboratory
University of Tennessee
CCDSC'14, Chateau des Contes

George Bosilca
Aurelien Bouteiller
Mathieu Faverge
Thomas Herault
Jack Dongarra

Programming Paradigms

- ✓PaRSEC offers a new prog. paradigm: PTG
  - ✓PTG: Parameterized Task Graph
- ✓What is wrong with current prog. paradigms?
- ✓What is the current programming paradigm?
  - ✓Pete says MPI & SPMD are not prog. paradigms

Serial Code

```
do i=0, len
     y(i) = y(i) + a*x(i)
enddo
```

OpenMP Code

```
#pragma omp parallel
do i=0, len
     y(i) = y(i) + a*x(i)
enddo
```

MPI Code

```
// ugly code here
do i=my_start, my_len
     y(i) = y(i) + a*x(i)
enddo
// more ugly code
```

MPI Code

```
// ugly code here
do i=my_start, my_len
     y(i) = y(i) + c(i)
enddo
// more ugly code
```

Data Distribution, parallelism and load balance are coupled.

MPI Code

```
// ugly code here
do  , start, my_len
     y(  (i) + c  (i)
enddo
// more ugly
```

No mechanism to deal with jitter.

Data Distribution, parallel nd load balance are coupled.

MPI+X Code

- Yeah, right.

MPI Programs are …

Coarse Grain Parallelism with Explicit Message Passing

Programs are essentially sequential, maybe with some code to coordinate.

- Vaidy Sunderam

Tile Algorithms

PARALLEL LINEAR ALGEBRA SOFTWARE FOR MULTICORE ARCHITECTURES

PLASMA

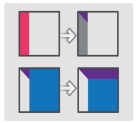
INNOVATIVE COMPUTING LABORATORY
THE UNIVERSITY of TENNESSEE

THE PARALLEL LINEAR ALGEBRA SOFTWARE FOR MULTICORE ARCHITECTURES (PLASMA) PROJECT aims to address the critical and highly disruptive situation that is facing the Linear Algebra and High Performance Computing community due to the introduction of multicore architectures. PLASMA's ultimate goal is to create software frameworks that enable programmers to simplify the process of developing applications that can achieve both high performance and portability across a range of new architectures. PLASMA uses a programming model that allows asynchronous, out-of-order scheduling of operations in order to achieve a scalable yet highly efficient software framework for Computational Linear Algebra applications.

TILE ALGORITHMS

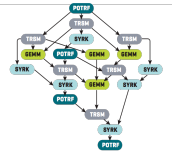
Unlike LAPACK, which uses block algorithms, PLASMA relies on tile algorithms to enable the use of fine grained parallelism.

LAPACK
Block algorithms



PLASMA
Tile algorithms

Tile algorithms of Linear Algebra operations can be represented as Directed Acyclic Graphs (DAG) where nodes represent the tasks in which the operation can be decomposed and the edges represent the dependencies among them. As long as the task execution order does not violate the dependencies, the result will be correct.



Example of a DAG for a Cholesky Factorization

PLASMA 2.1.0

- Solution of Linear Equations
- Linear Least Squares Problems
- Multiple Precision Support
- Mixed-Precision Iterative Solver
- Static Scheduling
- LAPACK Interface / Native Interface
- LAPACK-Compliant Error Handling
- LAPACK-Derived Testing Suite
- Thread Safety
- Windows, Linux, AIX, Mac OS
- PLASMA Users' Guide

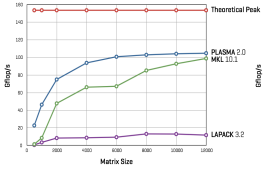
CURRENT RESEARCH

- Singular Value Decomposition
- Symmetric and Non Symmetric Eigenvalue Problems
- Dynamic Scheduling
- Communication Avoiding Algorithms
- Autotuning
- Distributed Memory Machines
- Hardware Accelerators

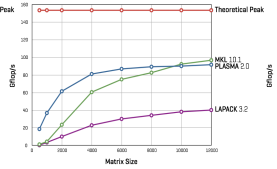
PERFORMANCE RESULTS DOUBLE PRECISION

CPU Intel Xeon 2.4 GHz Quad-socket Quad cores (16 cores total)

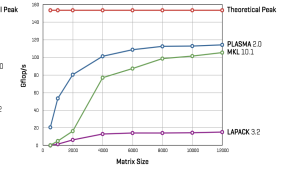
QR Factorization



LU Factorization



Cholesky Factorization



DOWNLOAD THE LIBRARY AT http://icl.eecs.utk.edu/plasma/

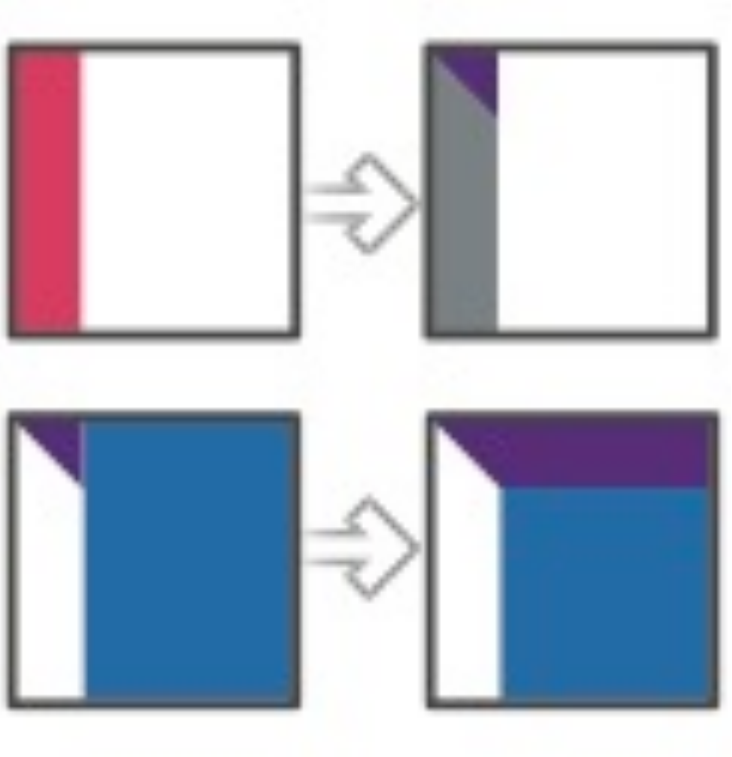
A COLLABORATION OF
The University of Tennessee · Berkeley · University of Colorado Denver

WITH SUPPORT FROM
The MathWorks · Microsoft

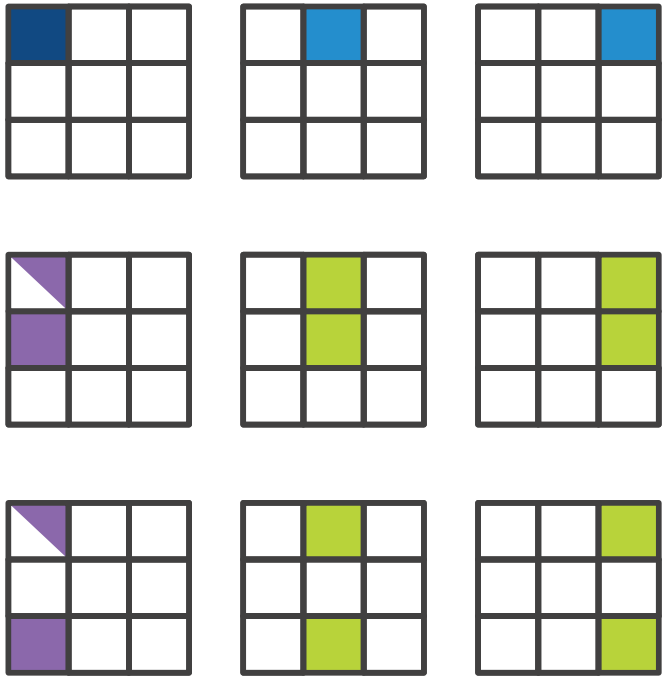
SPONSORED BY
NSF · CITR Center for Information Technology Research

Panel vs Tile Algorithms

LAPACK / Panel



PLASMA / Tile



What does the code look like?

```
for (k = 0; k < MT; k++) {
    Insert_Task( zgeqrt,  A[k][k], INOUT,  T[k][k], OUTPUT);
    for (m = k+1; m < MT; m++) {
        Insert_Task( ztsqrt,  A[k][k], INOUT | REGION_D|REGION_U,
                              A[m][k], INOUT | LOCALITY,
                              T[m][k], OUTPUT);
    }
    for (n = k+1; n < NT; n++) {
        Insert_Task( zunmqr,  A[k][k], INPUT | REGION_L,
                              T[k][k], INPUT,
                              A[k][m], INOUT);
        for (m = k+1; m < MT; m++) {
            Insert_Task( ztsmqr, A[k][n], INOUT,
                                 A[m][n], INOUT | LOCALITY,
                                 A[m][k], INPUT,
                                 T[m][k], INPUT);
```

What's wrong with this code

✗ It has:

- ✗ Control Flow
- ✗ Hints for runtime to infer Data Flow
- ✗ High memory requirements or reduced parallelism

✓ It should have:

- ✓ No (or minimal, user defined) Control Flow
- ✓ Explicit Data Flow
- ✓ Unhindered parallelism

Parameterized Task Graph

- Compressed form of the Execution DAG
- Fixed size (problem size independent)
- Task Classes w/ parameters
  - geqrt(k), tsqrt(k,m), unmqr(k,n), tsmqr(k,n,m)
- Precedence constraints between Tasks

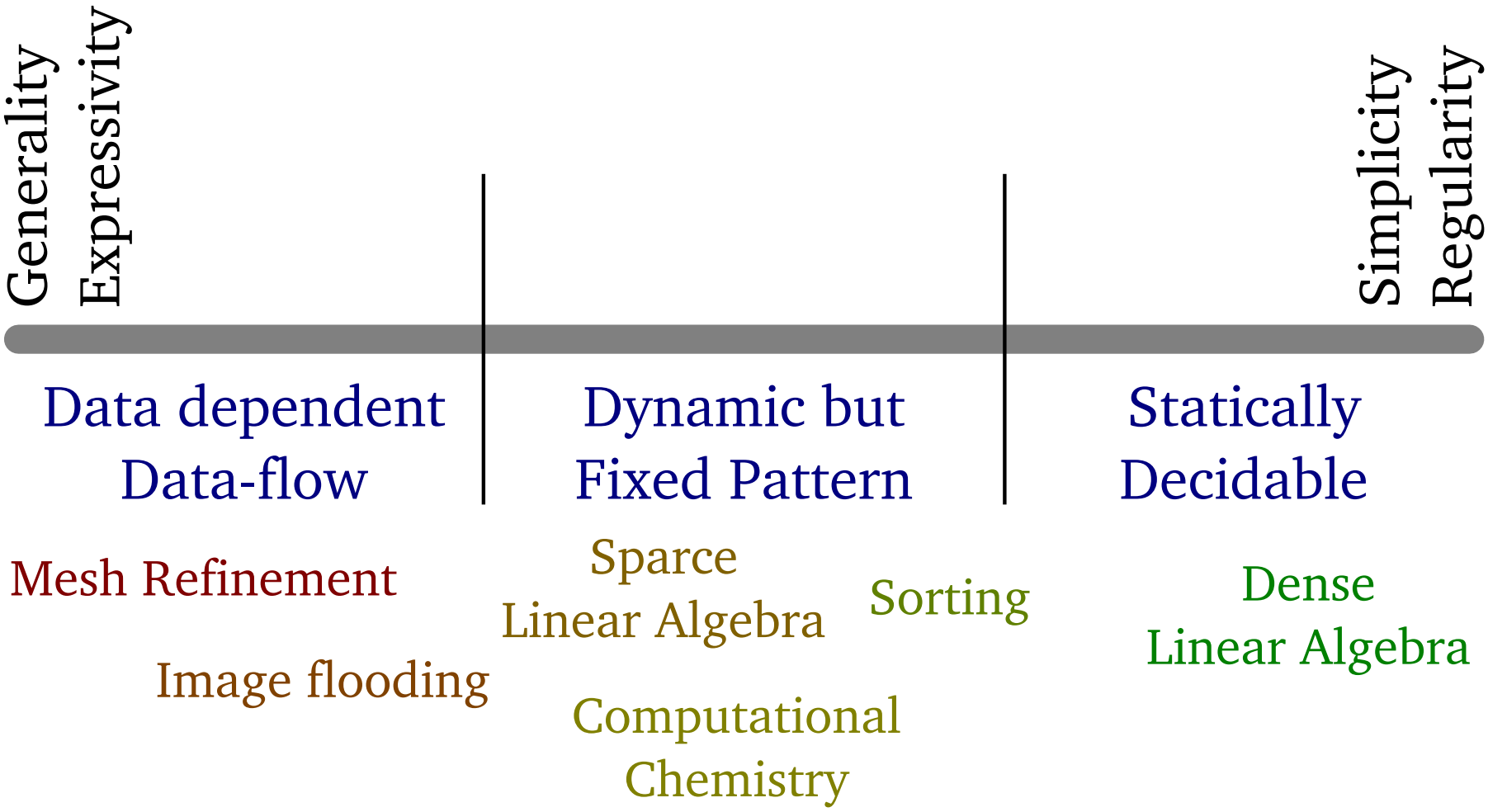
PTG: PING-PONG

```
PING(s)
 s = 0..max_steps-1
 : A(s)
 RW    A0 <- A(s)
          -> A0 PONG(s)
 READ A1 <- (s != 0) ? PONG(s-1)
BODY verify_response(A0, A1); END

PONG(s)
 s = 0..max_steps-2
 : A(s+1)
 RW    A0 <- A0 PING(s)
          -> A1 PING(s+1)
BODY /* do nothing on data */ END
```

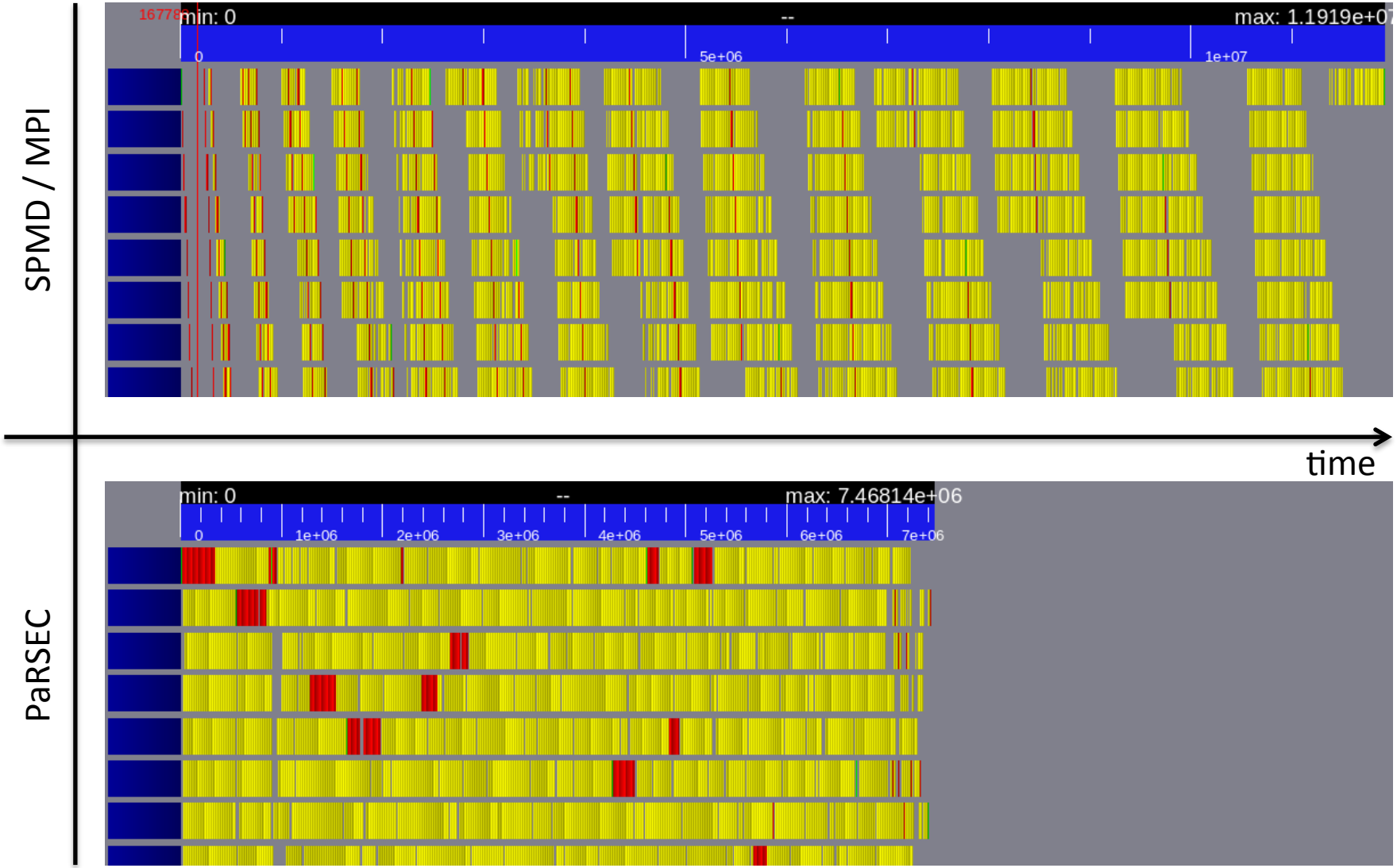
PTG: Binary Tree Reduction

```
BT_REDUC(tree, step, i)
  tree_count = count_bits(NT)
  tree  = 1 .. tree_count
  max_step = log_of_tree_size(NT, tree)
  step  = 1 .. max_step
  i   = 0 ..(1<<(max_step-step))-1
  offset = compute_offset(NT, tree)

  : dataA(offset+i*2,0)

  READ A   <- (1==step) ? A REDUCTION(offset+i*2)
           <- (1!=step) ? B BT_REDUC(tree,step-1,i*2)
  RW   B   <- (1==step)                          ? A REDUCTION(offset+i*2+1)
           <- (1!=step)                          ? B BT_REDUC(tree, step-1, i*2+1)
           -> ((max_step!=step) && (0==i%2)) ? A BT_REDUC(tree, step+1, i/2)
           -> ((max_step!=step) && (0!=i%2)) ? B BT_REDUC(tree, step+1, i/2)
           ->  (max_step==step)                  ? C LINEAR_REDUC(tree)
BODY int j; for(j=0; j<NB; j++){ REDUCE( A, B, j ); } END
```

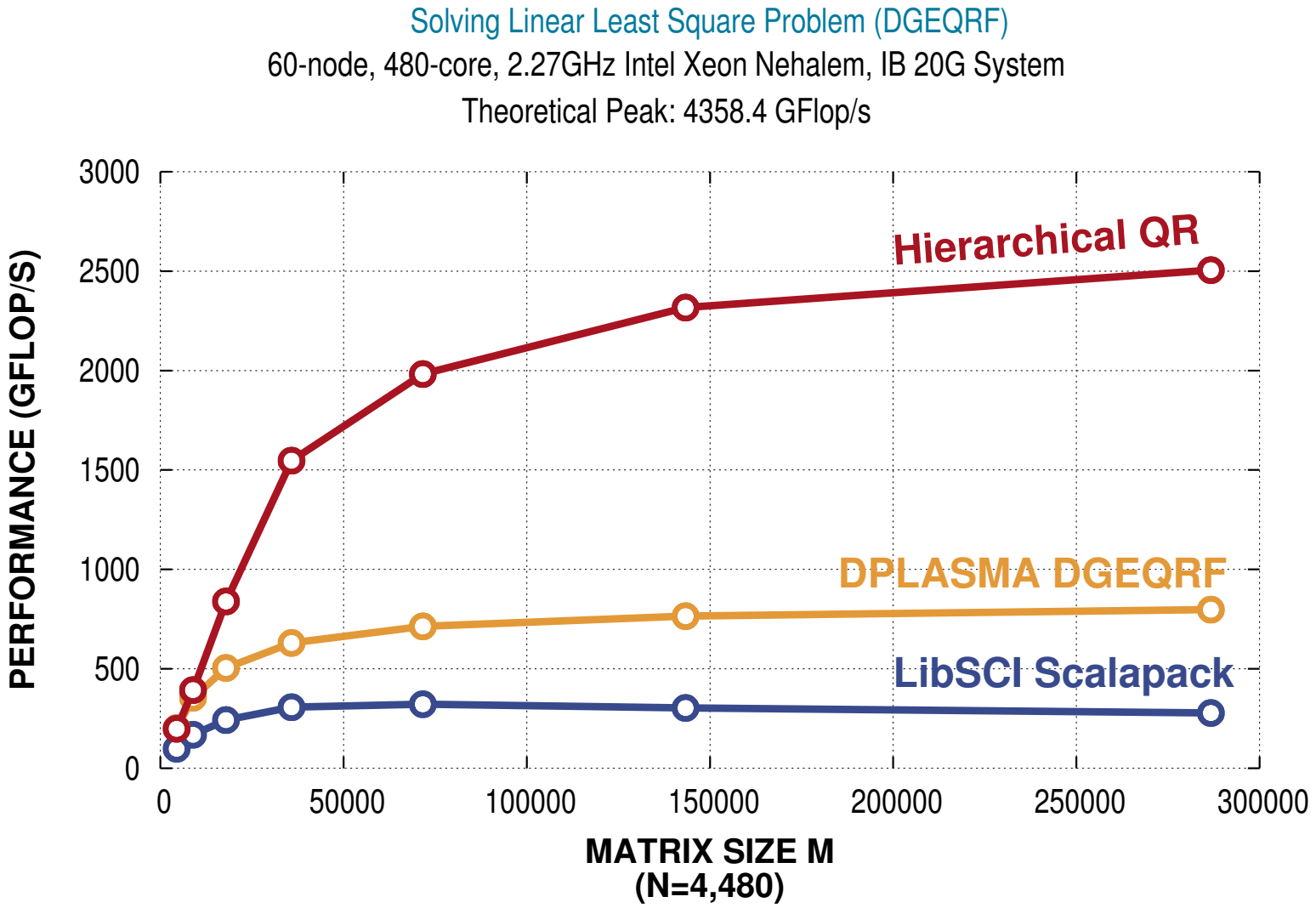
Who is this for?



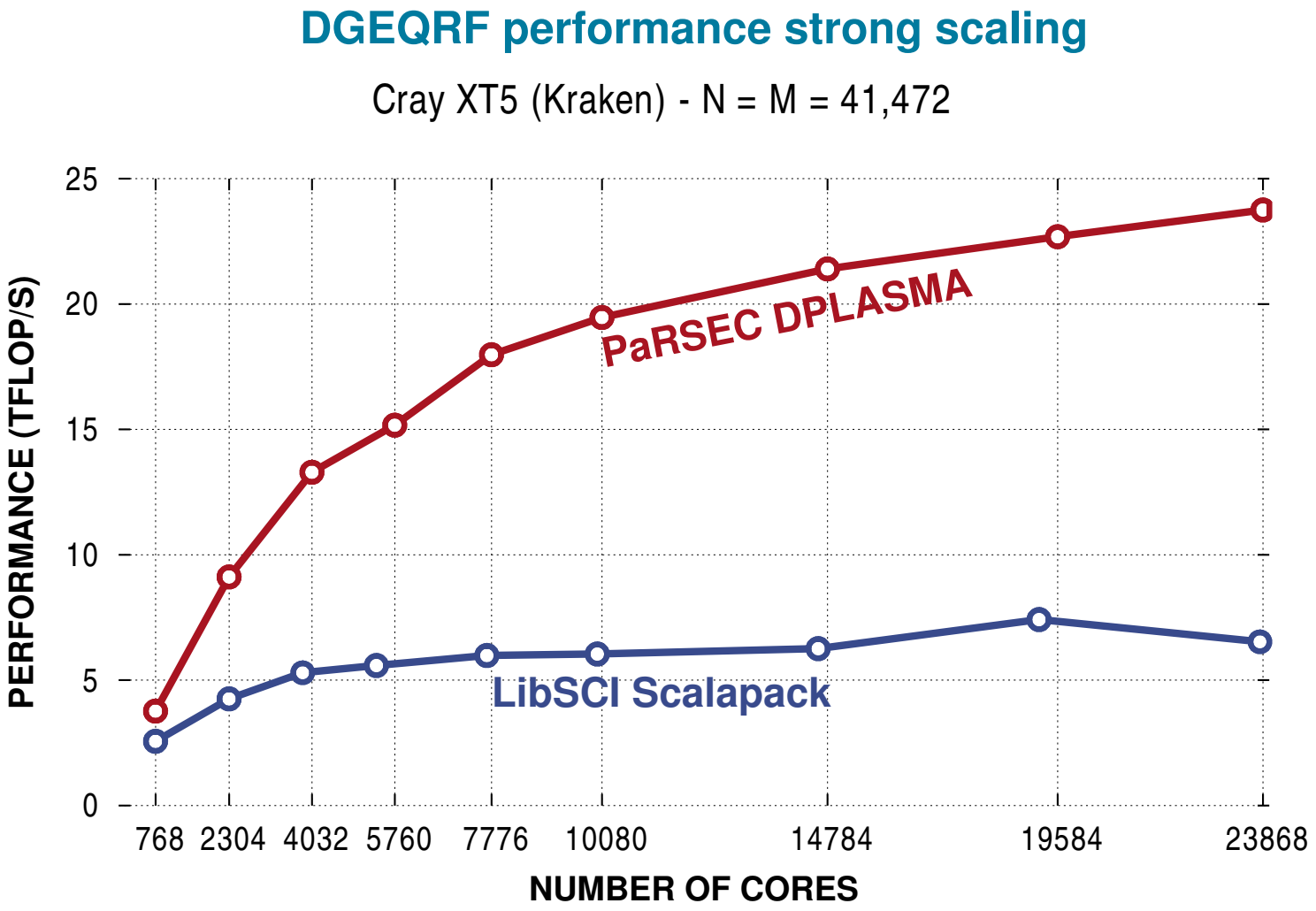
Load Balance, Idle time & Jitter



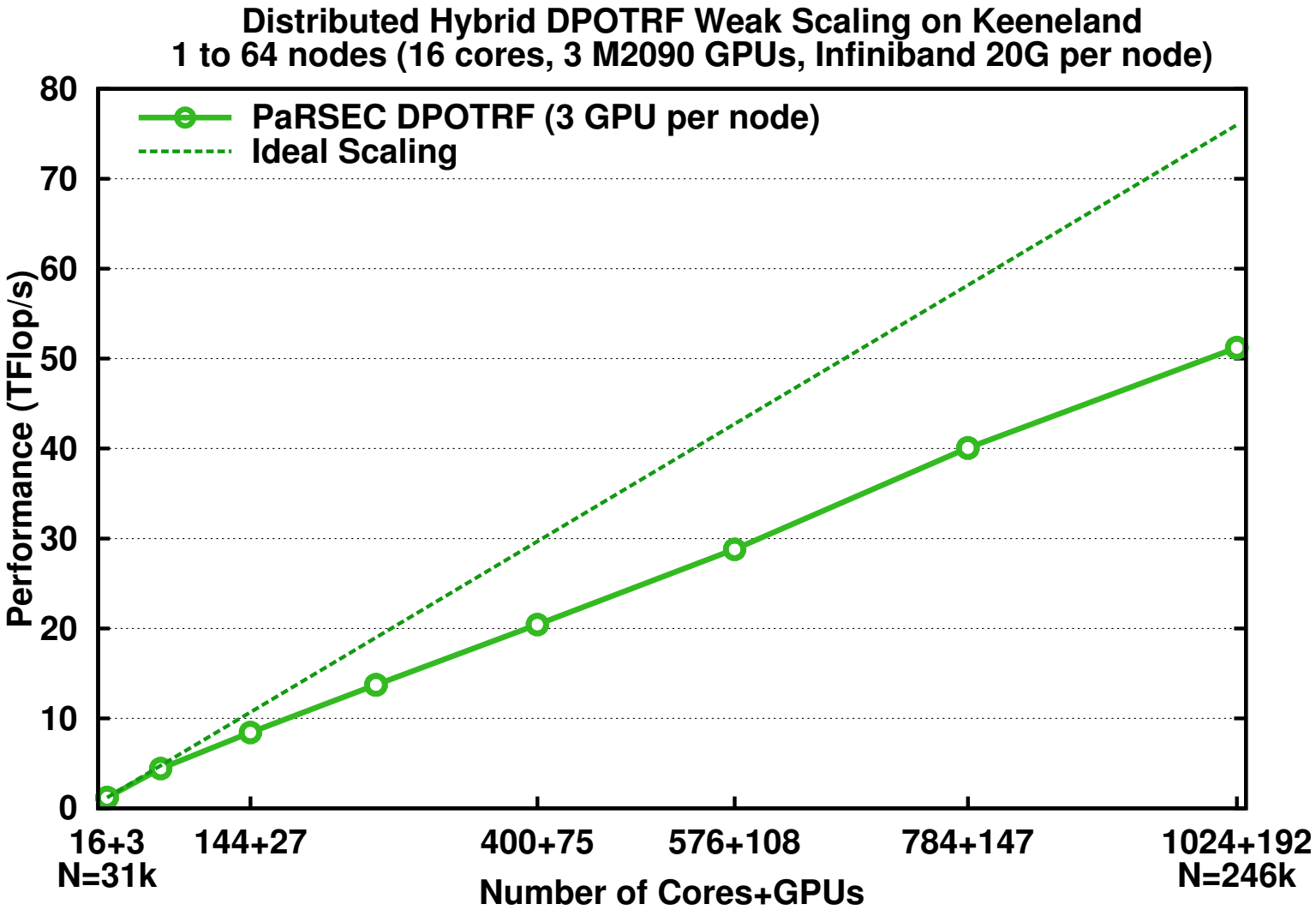
Performance: (H)QR



Performance: Systolic QR



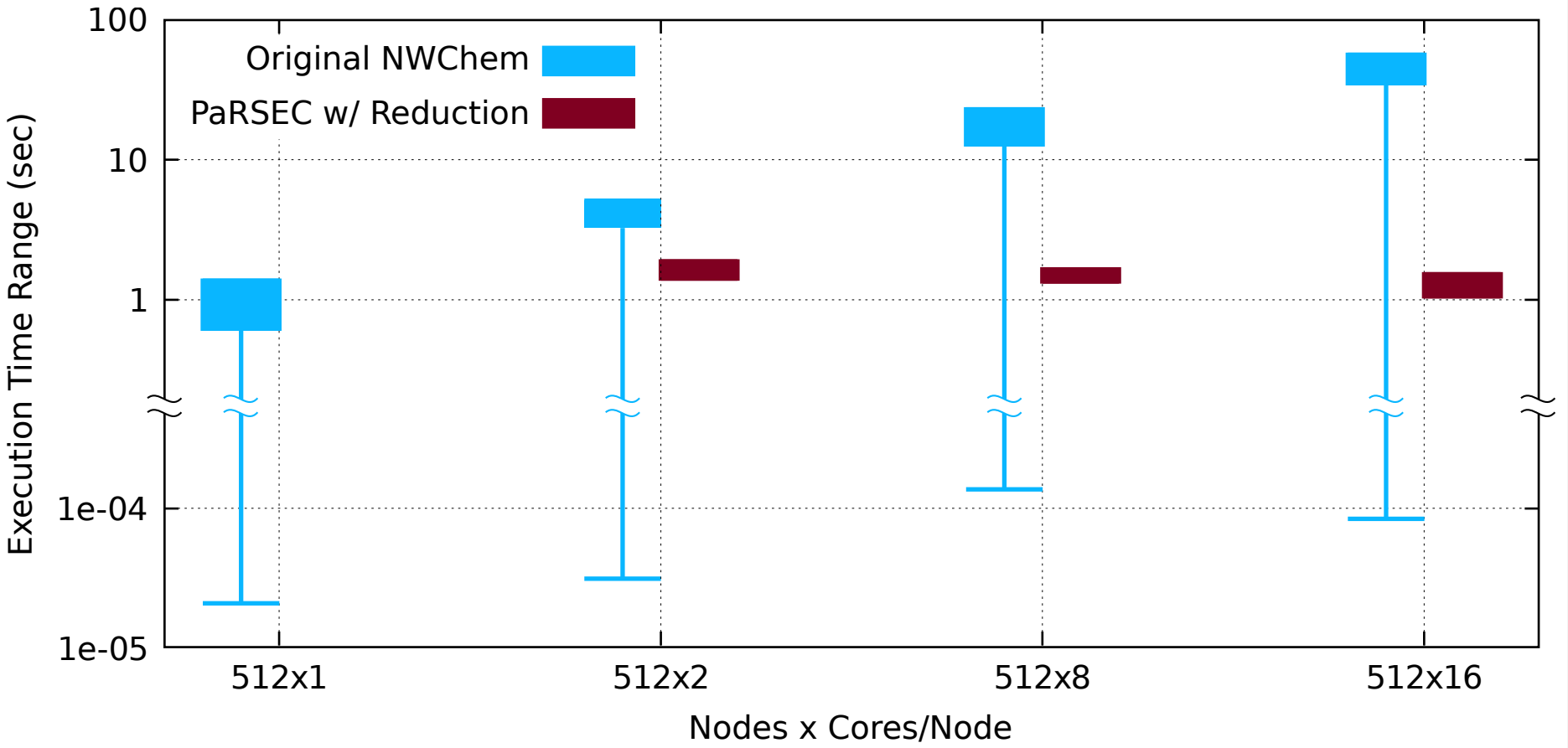
Distributed CPUs + GPUs



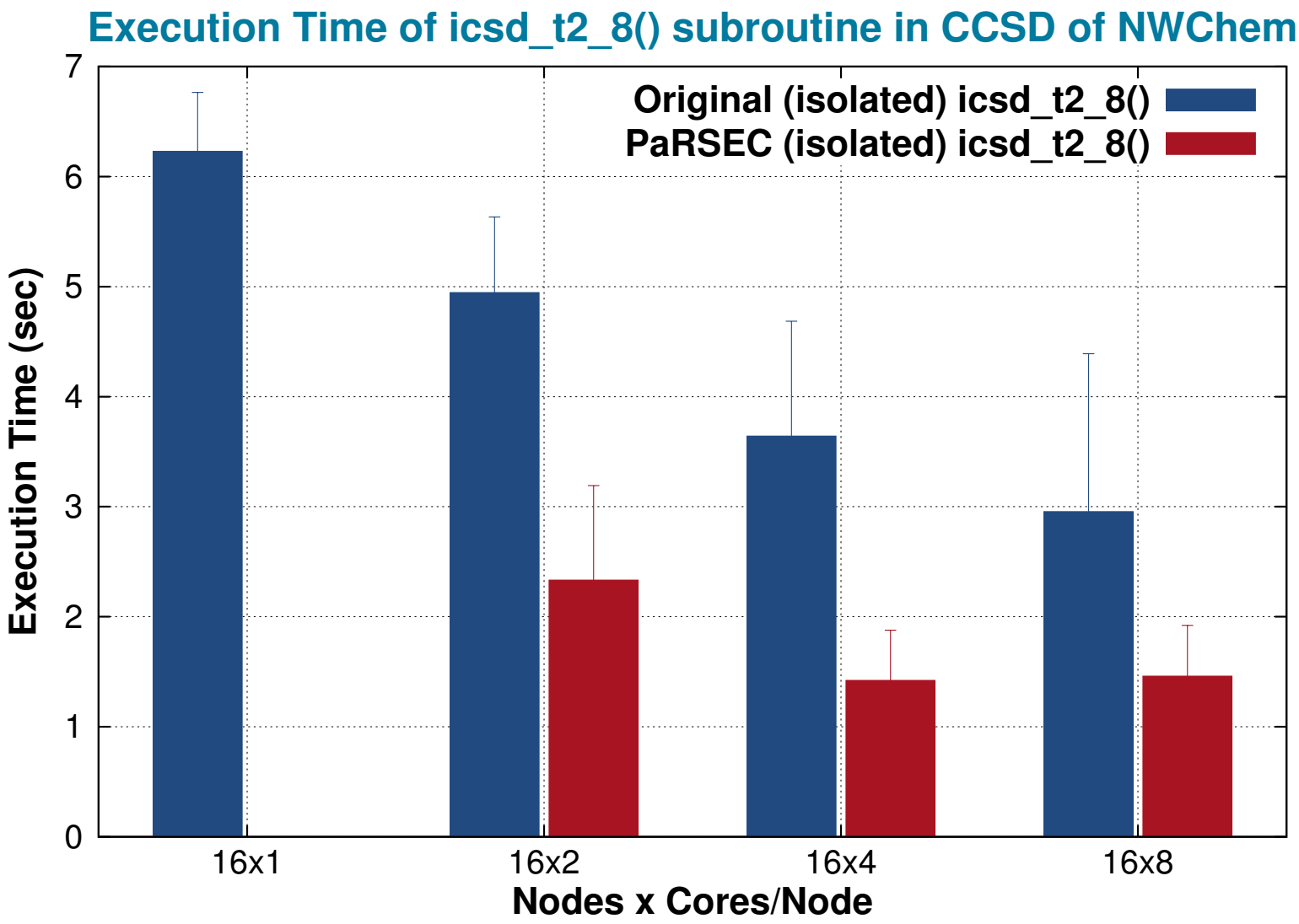
NWChem Coupled Cluster (CC)

- Computational Chemistry
- TCE Coupled Cluster
- Machine generated (awful) Fortran 77 code
- Behavior depends on dynamic, **immutable** data
- Long sequential chains of DGEMMs
- Work (chain) stealing through GA atomics
- Did I mention that the chains are sequential?

CC t1_2_2_2


Original NWChem
PaRSEC w/ Reduction
Execution Time Range (sec)
100
10
1
1e-04
1e-05
512x1
512x2
512x8
512x16
Nodes x Cores/Node

CC t2_8


Execution Time of icsd_t2_8() subroutine in CCSD of NWChem
Original (isolated) icsd_t2_8()
PaRSEC (isolated) icsd_t2_8()
Execution Time (sec)
0
1
2
3
4
5
6
7
16x1
16x2
16x4
16x8
Nodes x Cores/Node

Breaking the chains

```
DO p3b, p4b, h1b, h2b
      CALL DFILL(dimc,0.0d0,dbl_mb(k_c_sort),1)

      DO p5b, p6b
          IF (int_mb(k_spin+p5b-1) .eq. ...) THEN
              CALL DGEMM( ... )
          END IF
      END DO

      CALL TCE_SORT_4(dbl_mb(k_c_sort),dbl_mb(k_c), ... )
      CALL ADD_HASH_BLOCK(d_c,dbl_mb(k_c),dimc, expr )
END DO
```

Breaking the chains

```
DO p3b, p4b, h1b, h2b
      CALL DFILL(dimc,0.0d0,dbl_mb(k_c_sort),1)

      DO p5b, p6b
        IF (int_mb(k_spin+p5b-1) .eq. ...) THEN
          CALL DGEMM( ... )
        END IF
      ENDDO

      CALL TCE_SORT_4(dbl_mb(k_c_sort),dbl_mb(k_c), ... )
      CALL ADD_HASH_BLOCK(d_c,dbl_mb(k_c),dimc, expr )
END DO
```

DO

C = C + A*B

ENDDO

Breaking the chains

```
DO p3b, p4b, h1b, h2b
      CALL DFILL(dimc,0.0d0,dbl_mb(k_c_sort),1)

      DOANY
         C = C + A*B
      ENDDO

      CALL TCE_SORT_4(dbl_mb(k_c_sort),dbl_mb(k_c), ... )
      CALL ADD_HASH_BLOCK(d_c,dbl_mb(k_c),dimc, expr )
END DO
```

Conclusions

- PTG offers a Data-Flow based Prog. Paradigm
- PaRSEC offers state-of-the-art performance
- Data distribution is decoupled from Algorithm
- And you can bring your GPUs too
- And you can even bring your real applications
- CGP (MPI/SPMD) != highest performance
- CGP != easiest to program (?)