Do we need dataflow programming?

Anthony Danalis

Innovative Computing Laboratory

University of Tennessee

CCDSC'16, Chateau des Contes

Programming vs Execution

- ✓Dataflow based execution
  - ✓Think ILP, Out of order execution
  - ✓Automatically derived by hardware/compiler/etc
- ✓Dataflow programming
  - ✓Think Workflows
  - ✓Flow of data **explicitly** specified by human

Task-based vs Dataflow-based

Is task execution the same thing?

OpenMP

StarPU

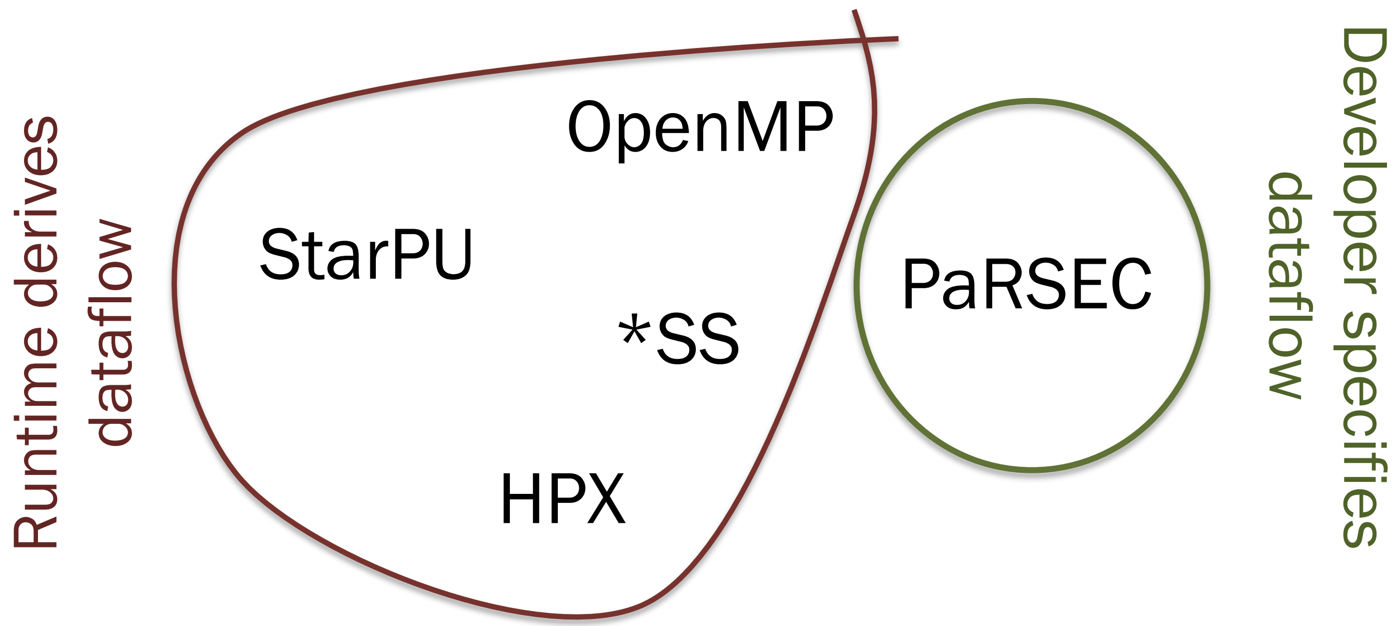
PaRSEC

*SS

HPX

Task-based vs Dataflow-based

Is task execution the same thing?



Limits of deriving the dataflow

```
P: nodes
N: number of kernel executions
Tk: kernel execution time
To: overhead of discovery

To*N << Tk*N/P =>
To*N <= 0.1*Tk*N/P =>
P <= 0.1*Tk/To

To = 100ns, Tk = 100us => P <= 100
```

Explicit Dataflow Programming

Why does Explicit Dataflow Programming (EDP) differ from everything else?

The human developer explicitly expresses the semantics of the algorithm/application in a way that the runtimes/compilers can directly take advantage of without deriving information.

Explicit Dataflow Programming

Why does Explicit Dataflow Programming (EDP) differ from everything else?

The human developer explicitly expresses the semantics of the algorithm/application in a way that the runtimes/compilers can directly take advantage of without deriving information.

Benefits:

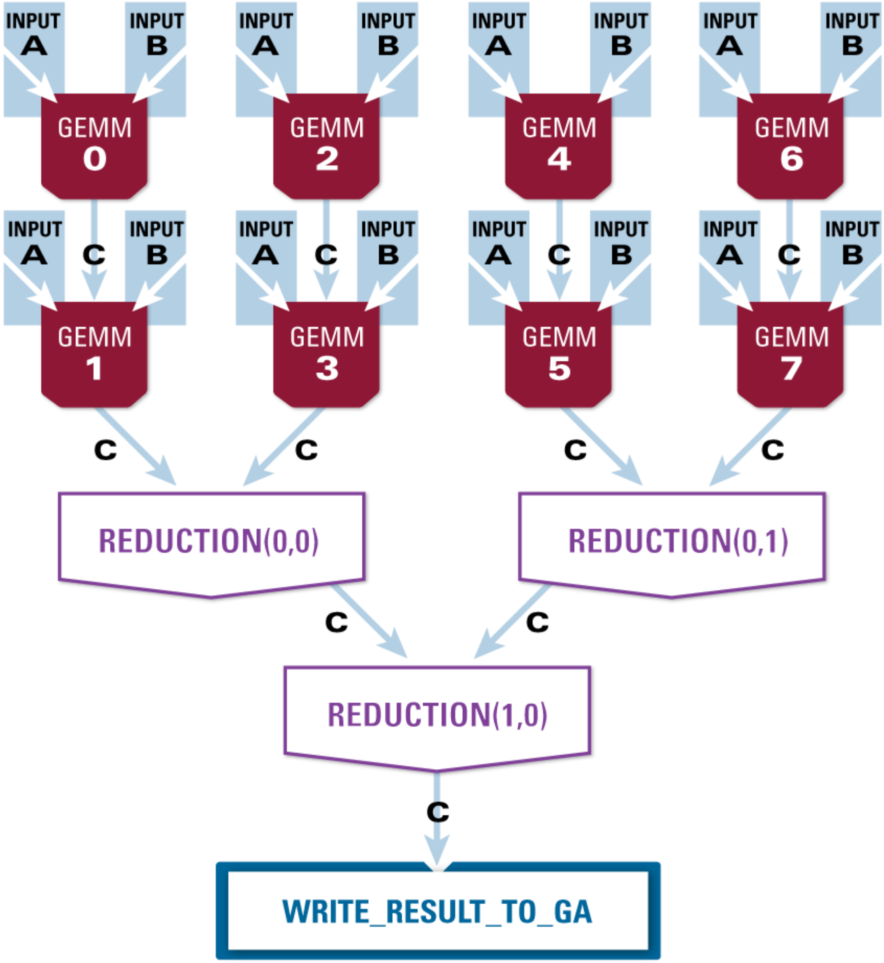
Perfect Parallelism, Automatic Comm./Comp. overlap, Collective operation detection.

Perf. Case study: NWChem CCSD

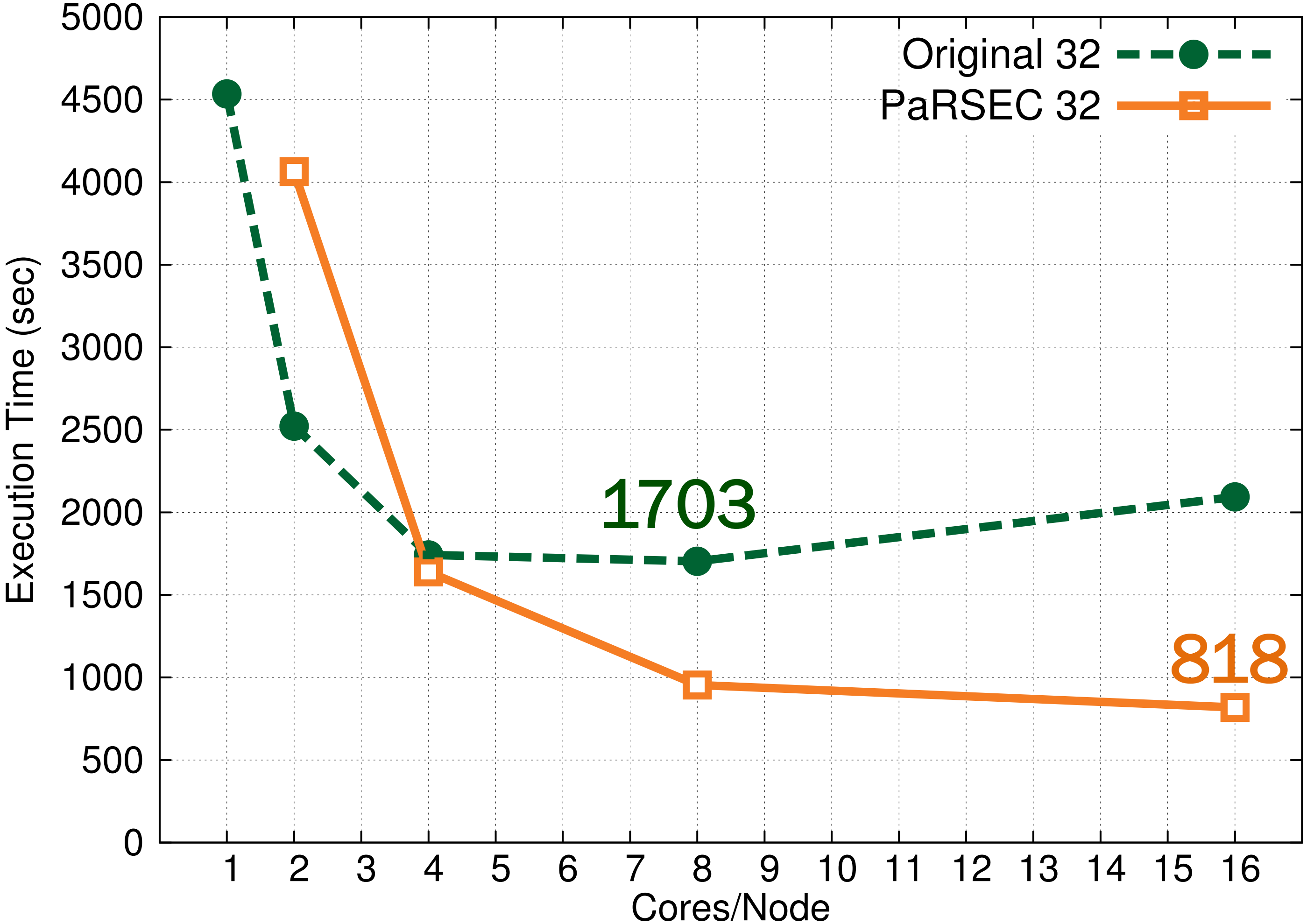
```
DO {x4}
  CALL nxt_ctx_next(ctx, icounter, next)          Global work stealing
  IF ( (int_mb(...)+...).ne.8 ) THEN
    CALL MA_PUSH_GET()                            Allocate and initialize C
    CALL DFILL()
    DO {x2}
      IF ( (int_mb(...)+... .eq. int_mb(...) ) THEN
        CALL MA_PUSH_GET(...,k_a)                 Allocate and fetch A
        CALL GET_HASH_BLOCK(d_a, dbl_mb(k_a), ...)  (same for B, not shown)

        CALL DGEMM(...)                           Actual work
      END IF
    END DO
    CALL TCE_SORT_4(dbl_mb(k_c), ...)
    CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)    Push C back
END DO
```

Structure of PTG computation



CCSD Execution Time on 32 nodes

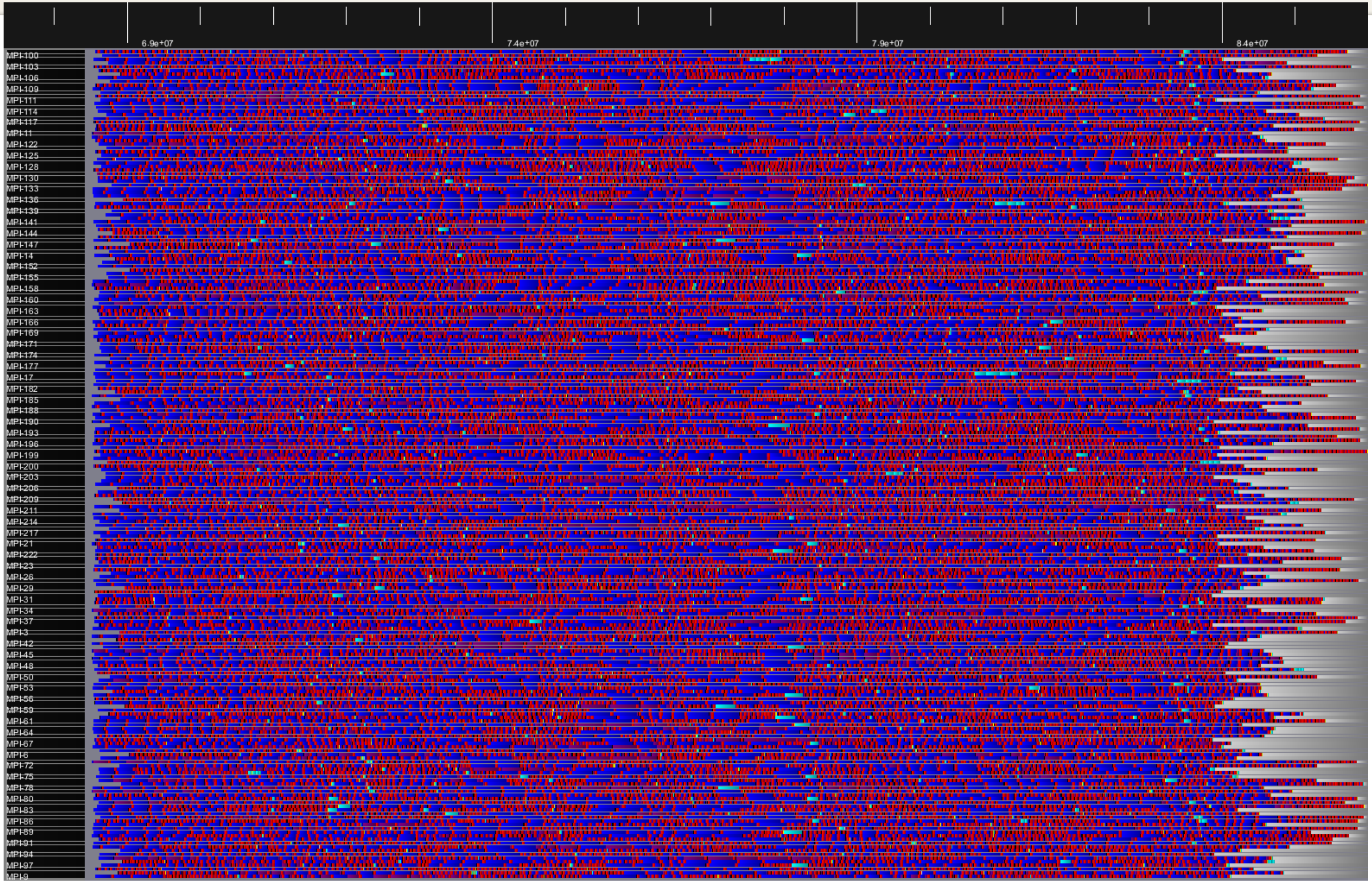


Performance bottlenecks

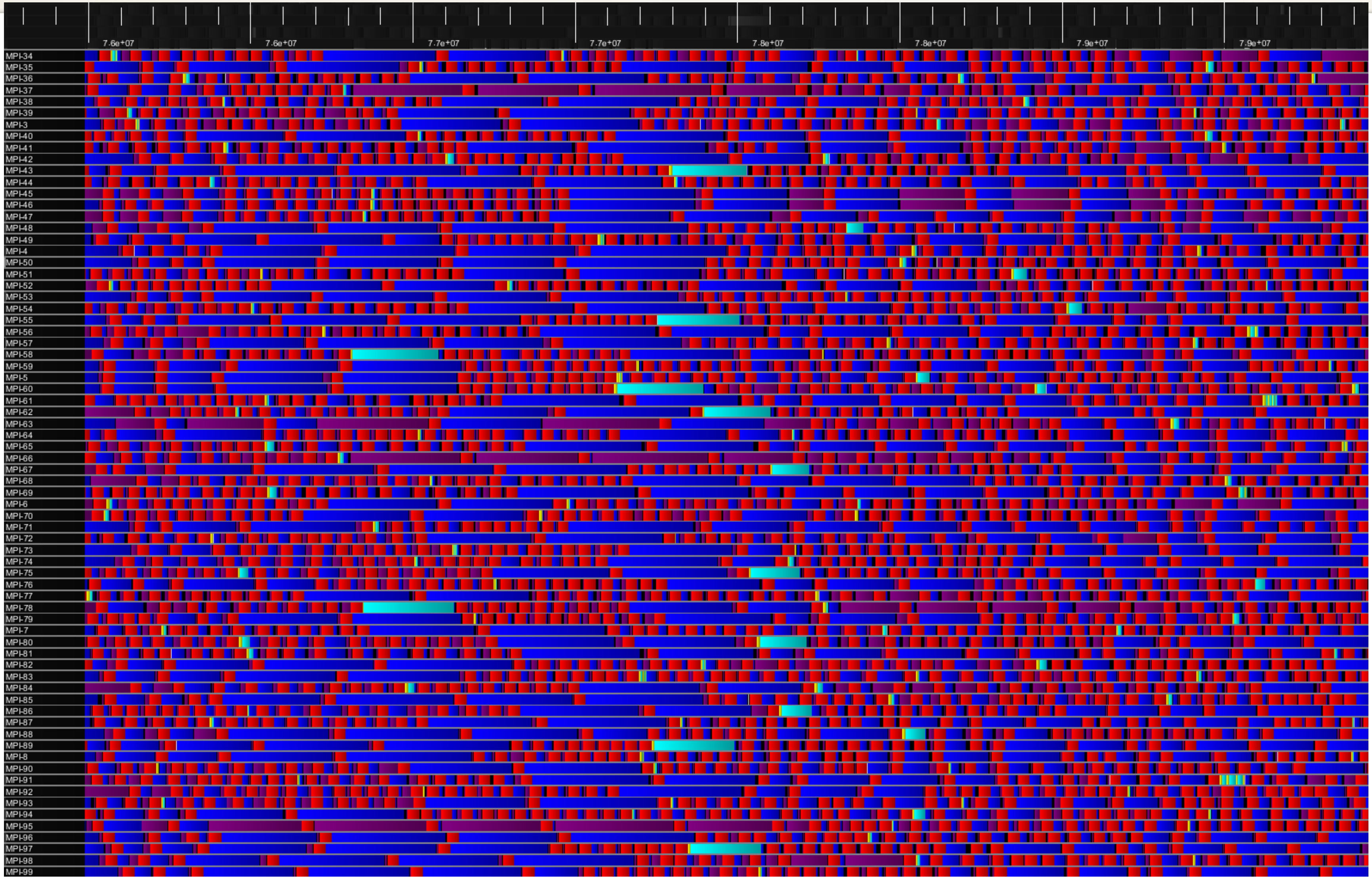
```
DO {x4}
 CALL nxt_ctx_next(ctx, icounter, next)
 IF ( (int_mb(...)+...).ne.8 ) THEN
  CALL MA_PUSH_GET()
  CALL DFILL()
  DO {x2}
    IF ( (int_mb(...)+... .eq. int_mb(...) ) THEN
      CALL MA_PUSH_GET(...,k_a)
      CALL GET_HASH_BLOCK(d_a, dbl_mb(k_a), ...)
      CALL DGEMM(...)
    END IF
  END DO
  CALL TCE_SORT_4(dbl_mb(k_c), ...)
  CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)
END DO
```

1. Global atomic
2. Coarse grain parallelism
3. No opportunity for comm/comp overlap

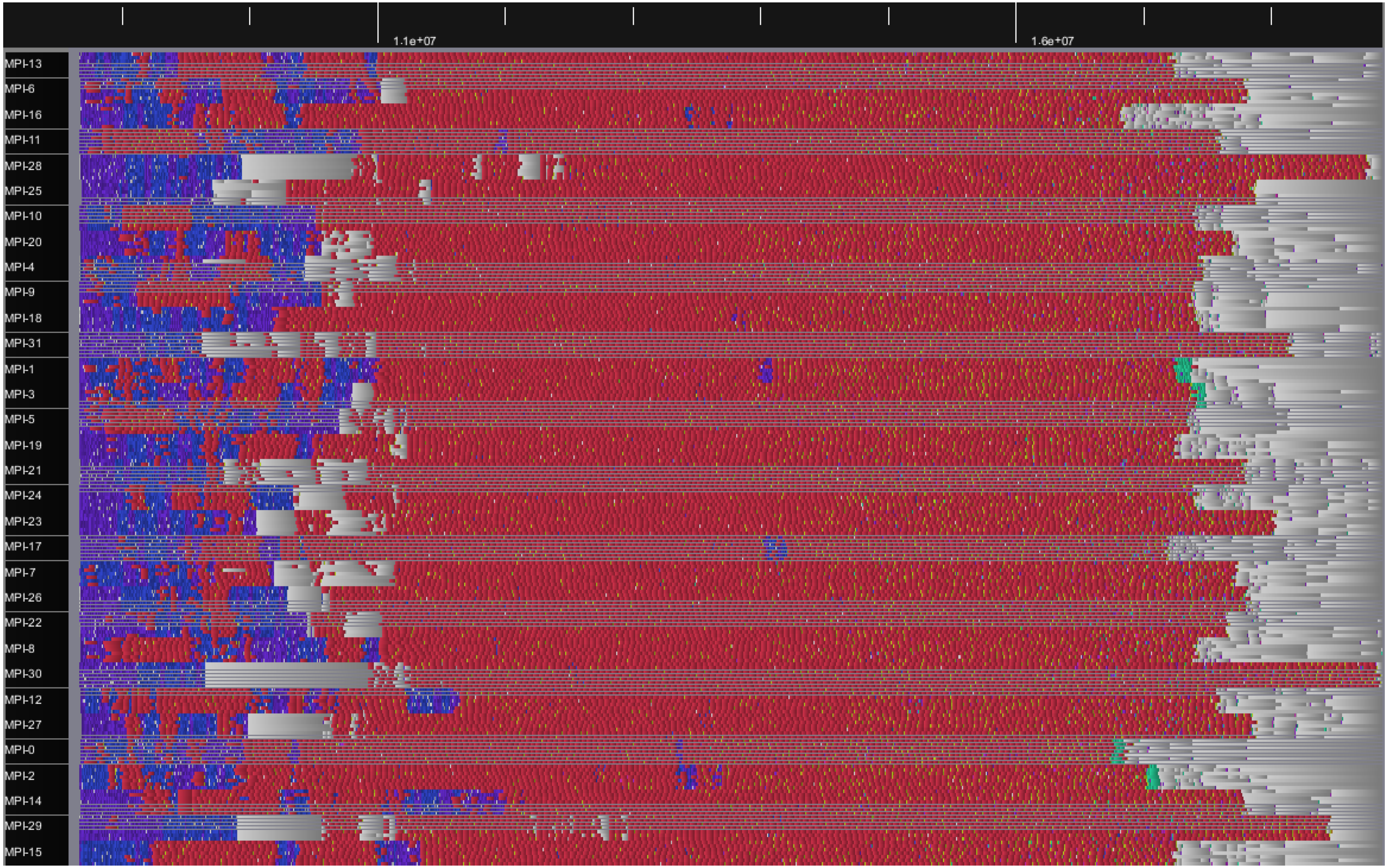
Trace of Original code



Trace of Original code (zoom)



Trace of PaRSEC implementation



Whose fault is the bad performance?

Audience participation. Choose who to blame:

- MPI
- Developers
- Programming paradigm (Coarse Grain Parallelism)
- Vetter (for not telling his users about dataflow)

Whose fault is it?

Audience participation. Choose who to blame:

- MPI
- Developers
- Programming paradigm (Coarse Grain Parallelism)
- Vetter (for not telling his users about dataflow)

MPI has a simple and an advanced API and many developers use only the simple one.
- Rusty

Message so far

- Using CGP does <u>not</u> scale
- Using Dataflow execution <u>does</u>
- BUT, developers have to understand their code

Sure, but can we make EDP easy?

Can we make dataflow <u>execution</u> harness all the benefits without explicit dataflow <u>programming</u>?

Sure, but can we make EDP easy?

Can we make dataflow <u>execution</u> harness all the benefits without explicit dataflow <u>programming</u>?

Yes, we can. In some cases. Maybe?

Bridging Explicit & Implicit dataflow

✓Reduce the cost of discovery

- Code specialization
  - Developer expertise
  - Results of compiler analysis

✓Harness benefits of parametric representation

- Compress the Graph on the fly
  - Detect patterns in series that translate to expressions, or functions
  - Use compiler inserted hints

Reduce the unnecessary discovery

```
DO {x4}
  CALL nxt_ctx_next(ctx, icounter, next)
  IF ( (int_mb(...)+...).ne.8 ) THEN
    CALL MA_PUSH_GET()
    CALL DFILL()
    DO {x2}
      IF ( (int_mb(...)+... .eq. int_mb(...) ) THEN
        CALL MA_PUSH_GET(...,k_a)
        CALL GET_HASH_BLOCK(d_a, dbl_mb(k_a), ...)
        CALL DGEMM(...)
      END IF
    END DO
    CALL TCE_SORT_4(dbl_mb(k_c), ...)
    CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)
END DO
```

Insert_Task

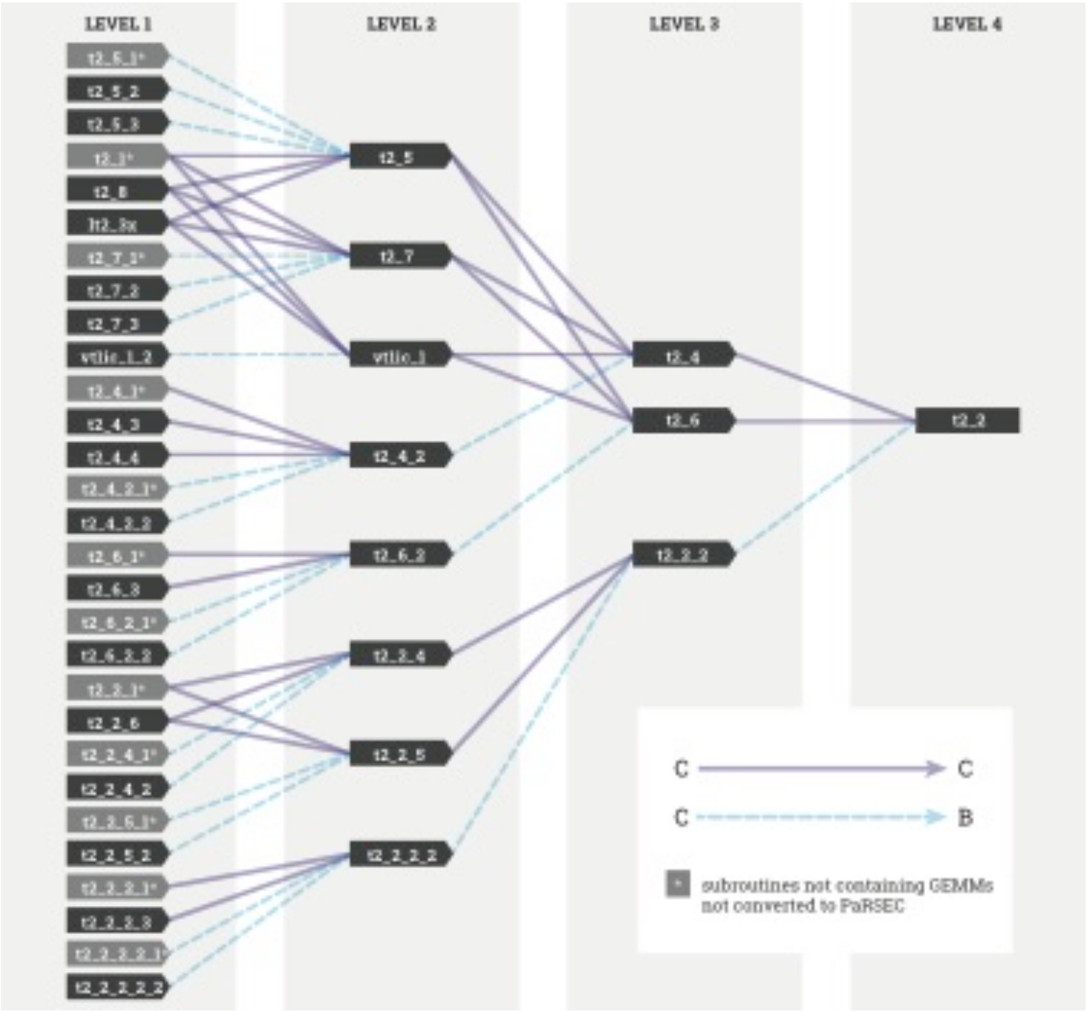
Reduce the unnecessary discovery

```
DO {x4}
  CALL nxt_ctx_next(ctx, icounter, next)
  IF ( (int_mb(...)+...).ne.8 ) THEN
    CALL MA_PUSH_GET()                          <- Handle Generation
    CALL DFILL()
    DO {x2}
      IF ( (int_mb(...)+... .eq. int_mb(...) ) THEN
        CALL MA_PUSH_GET(...,k_a)               <- Handle Generation
        CALL GET_HASH_BLOCK(d_a, dbl_mb(k_a), ...)  <- Data Fetching
        CALL DGEMM(...)                         <- Insert_Task
      END IF
    END DO
    CALL TCE_SORT_4(dbl_mb(k_c), ...)
    CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)  <- Data Flushing
END DO
```

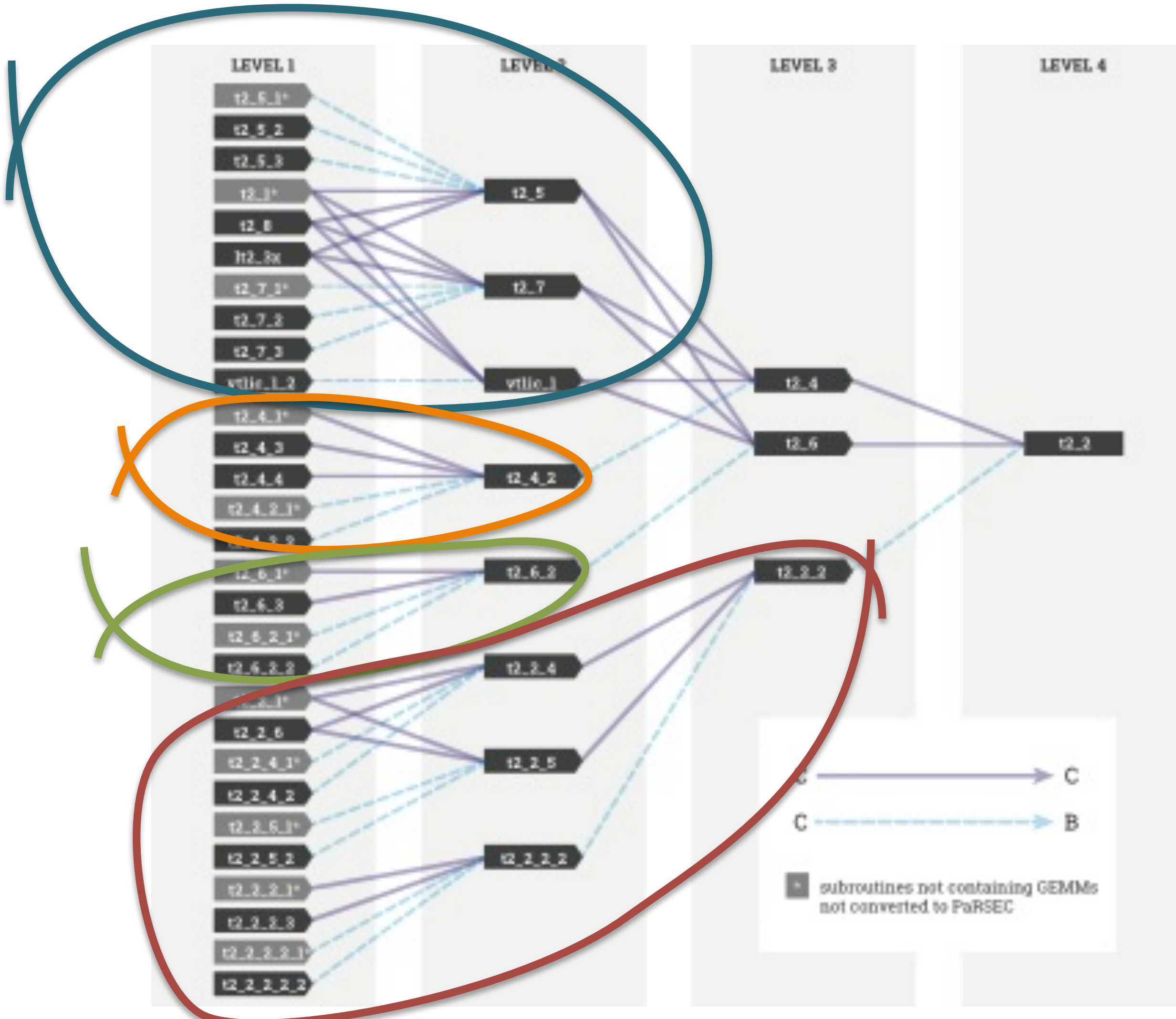
Reduce the unnecessary discovery

```
DO {x4}                                            Pleasantly Parallel
  CALL nxt_ctx_next(ctx, icounter, next)
  IF ( (int_mb(...)+...).ne.8 ) THEN
    CALL MA_PUSH_GET()                             Handle Generation
    CALL DFILL()
    DO {x2}
      IF ( (int_mb(...)+... .eq. int_mb(...) ) THEN
        CALL MA_PUSH_GET(...,k_a)                  Handle Generation
        CALL GET_HASH_BLOCK(d_a, dbl_mb(k_a), ...) Data Fetching
        CALL DGEMM(...)                            Insert_Task
      END IF
    END DO
    CALL TCE_SORT_4(dbl_mb(k_c), ...)
    CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)     Data Flushing
END DO
```

Dataflow between subroutines



Code grouping based on dataflow



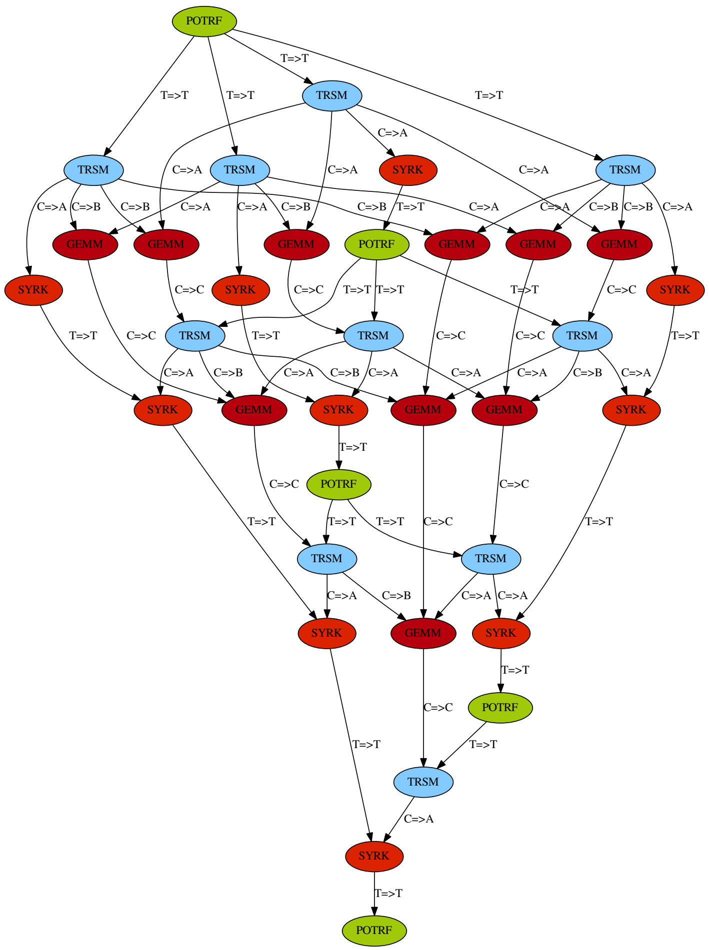
Message so far

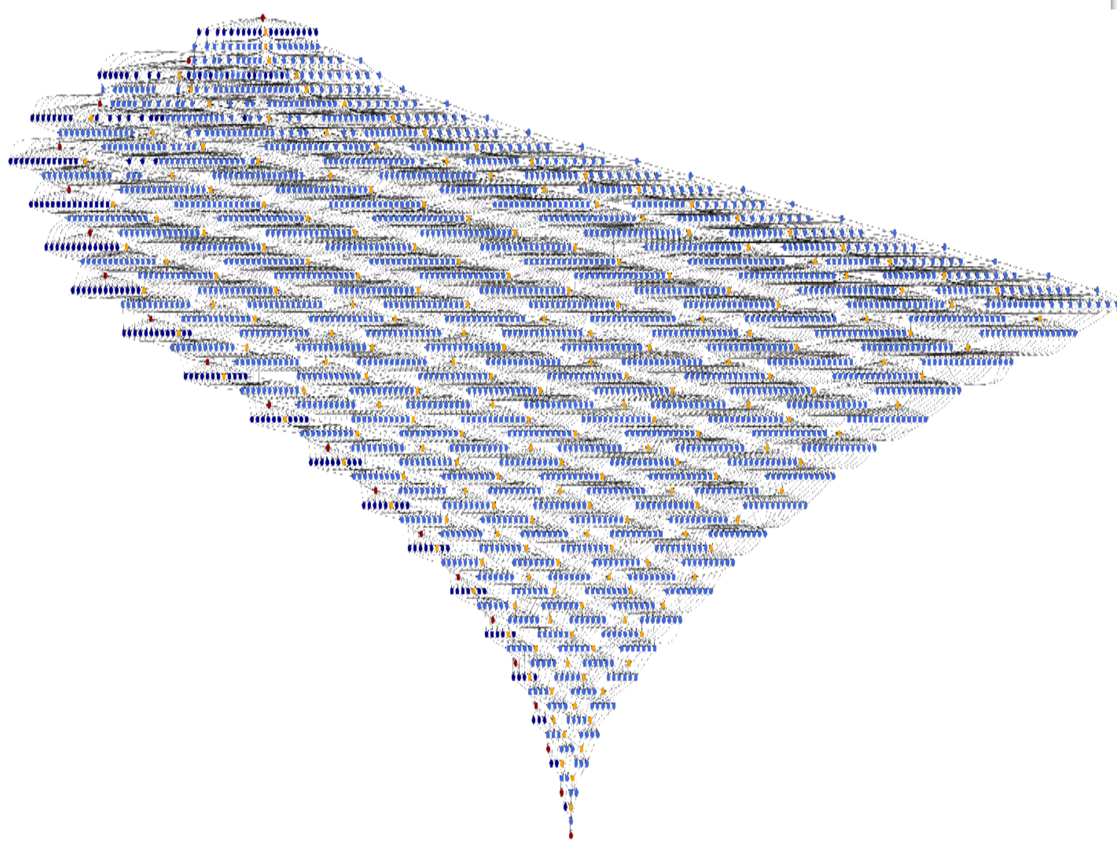
- Discovering the whole DAG does <u>not</u> scale
- Pruning the DAG requires human expertise
- Compiler analysis can assist with pruning
- BUT, developers have to understand their code

Compressing the DAG to a PTG?

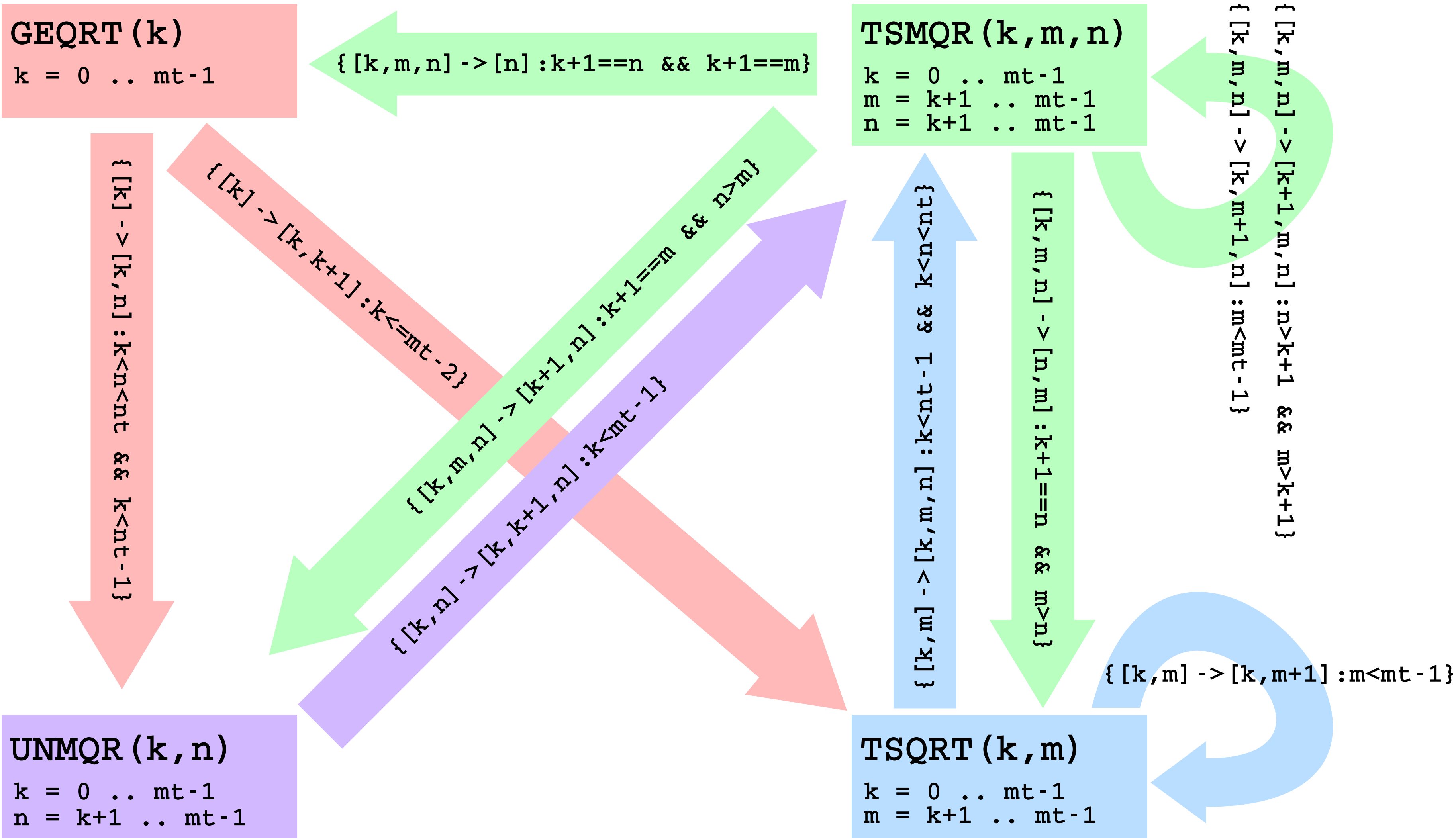
```
for (k = 0; k < MT; k++) {
    Insert_Task( zgeqrt,  A[k][k], INOUT,  T[k][k], OUTPUT);
    for (m = k+1; m < MT; m++) {
        Insert_Task( ztsqrt,  A[k][k], INOUT | REGION_D|REGION_U,
                              A[m][k], INOUT | LOCALITY,
                              T[m][k], OUTPUT);
    }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





Fully compressed DAG (PTG)



Compressing the DAG to a PTG?

```
for (k = 0; k < MT; k++) {
    Insert_Task( zgeqrt,  A[k][k],                     );
    for (m = k+1; m < MT; m++) {
        Insert_Task( ztsqrt,  A[k][                    GION_U,
                              A[m][
                              T[m][
    }
    for (n = k+1; n < NT; n++) {
        Insert_Task( zunmqr,  A[k][
                              T[k][
                              A[k][n], 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);
```

Task_A
Task_B
Task_B
Task_B
Task_C
Task_D
Task_D
Task_D
...

Compressing the DAG to a PTG?

```
for (k = 0; k < MT; k++) {
    Insert_Task( zgeqrt,  A[k][k], 
    for (m = k+1; m < MT; m++) {
        Insert_Task( ztsqrt,  A[k][                             GION_U,
                              A[m][
                              T[m][
    }
    for (n = k+1; n < NT; n++) {
        Insert_Task( zunmqr,  A[k][
                              T[k][
                              A[k][n], 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);
```

Task_A
Task_B
Task_B
Task_B
Task_C
Task_D
Task_D
Task_D
...

Compressing the DAG to a PTG?

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

GION_U,

);

Task_A
Task_B
Task_B
Task_B
Task_C
Task_D(1-3)
...

Compressing the DAG to a PTG?

```
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][n], 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);
```

Iteration_vector(k,n,m)
Indices(A[k][n],k,n)

Compressing the DAG to a PTG?

```
for (k = 0; k < MT; k++) {
    Insert_Task( zgeqrt,  A[k][k],                               );
    for (m = k+1; m < MT; m++) {
        Insert_Task( ztsqrt,  A[k][                   GION_U,
                              A[m][
                              T[m][
    }
    for (n = k+1; n < NT; n++) {
        Insert_Task( zunmqr,  A[k][
                              T[k][
                              A[k][
        for (m = k+1; m < MT; m++)
            Insert_Task( ztsmqr, A[
                                 A[                   ,
                                 A[
                                 T[
```

Task_A
Task_B
Task_B
Task_B
Task_C
Task_D
Task_D
Task_D
Task_C
Task_D
Task_D
Task_D
Task_A
...

Compressing the DAG to a PTG?

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

Task_A
Task_B
Task_B
Task_B
Task_C(k,n) k<m<MT
Task_D(k,n,m)
...

Conclusion

- ☺ Dataflow execution is more scalable than CGP
- ☺ Dataflow programming can maximize benefits
- ☹ Compilers cannot do it by themselves
  - ☹ Not even Torsten's compiler!
- ☹ Runtimes can, but at a cost

➢ Dataflow for the masses means sharing the load between developer, compiler and runtime.

Quotes

Developers know about their program much more than a compiler can ever figure out.

- Doug Miles

Let the human do what humans do best.

- Jeff Hollingsworth