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
- *SS
- PaRSEC
- HPX
Task-based vs Dataflow-based

Is task execution the same thing?

Runtime derives dataflow

Developer specifies dataflow

OpenMP

StarPU

*SS

PaRSEC

HPX
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()
    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),...)
      END IF
    END DO
    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

[Diagram showing the computation flow:
- INPUT A and B are fed into multiple GEMM operations (GEMM 0, GEMM 1, etc.).
- Each GEMM operation outputs a result, labeled C.
- The results from GEMM 1 and GEMM 3 are passed to REDUCTION(0,0).
- The results from GEMM 2 and GEMM 4 are passed to REDUCTION(0,1).
- The results from GEMM 5 and GEMM 6 are passed to REDUCTION(1,0).
- The final output is WRITE_RESULT_TO_GA.]
CCSD Execution Time on 32 nodes

![Graph showing CCSD execution time on 32 nodes with two lines, one for Original 32 and one for PaRSEC 32. The graph plots execution time (in seconds) against cores per node. At 1 core per node, the Original 32 line is at 818 seconds and the PaRSEC 32 line is at 1703 seconds. At 16 cores per node, the Original 32 line is at 5000 seconds and the PaRSEC 32 line is at 818 seconds.](image)
Performance bottlenecks

1. Global atomic
2. Coarse grain parallelism
3. No opportunity for comm/comp overlap
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
• Using CGP does not scale
• Using Dataflow execution does
• BUT, developers have to understand their code
Sure, but can we make EDP easy?

Can we make dataflow execution harness all the benefits without explicit dataflow programming?
Sure, but can we make EDP easy?

Can we make dataflow execution harness all the benefits without explicit dataflow programming?

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()
    END DO
    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
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
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
END DO
CALL TCE_SORT_4(dbl_mb(k_c), ...)
CALL ADD_HASH_BLOCK(d_c, dbl_mb(k_c), ...)
END DO
Dataflow between subroutines
Code grouping based on dataflow
Message so far

• Discovering the whole DAG does not scale
• Pruning the DAG requires human expertise
• Compiler analysis can assist with pruning
• BUT, developers have to understand their code
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)

**GEQRT(k)**
\[ k = 0 \ldots mt-1 \]

**UNMQR(k,n)**
\[ k = 0 \ldots mt-1 \]
\[ n = k+1 \ldots mt-1 \]

**TSMQR(k,m,n)**
\[ k = 0 \ldots mt-1 \]
\[ m = k+1 \ldots mt-1 \]
\[ n = k+1 \ldots mt-1 \]

**TSQRT(k,m)**
\[ k = 0 \ldots mt-1 \]
\[ m = k+1 \ldots mt-1 \]
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], T[m][k], LOCALITY);
    }
    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);
        }
    }
}
Compressing the DAG to a PTG?

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

Compressing the DAG to a PTG?

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

```
  Task_A
  Task_B
  Task_B
  Task_B
  Task_B
  Task_C
dataflow
  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)
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[k][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[k][n], INOUT | LOCALITY, A[m][k], INPUT,
                        T[m][k], INPUT);
        }
    }
}
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);
    }
}
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