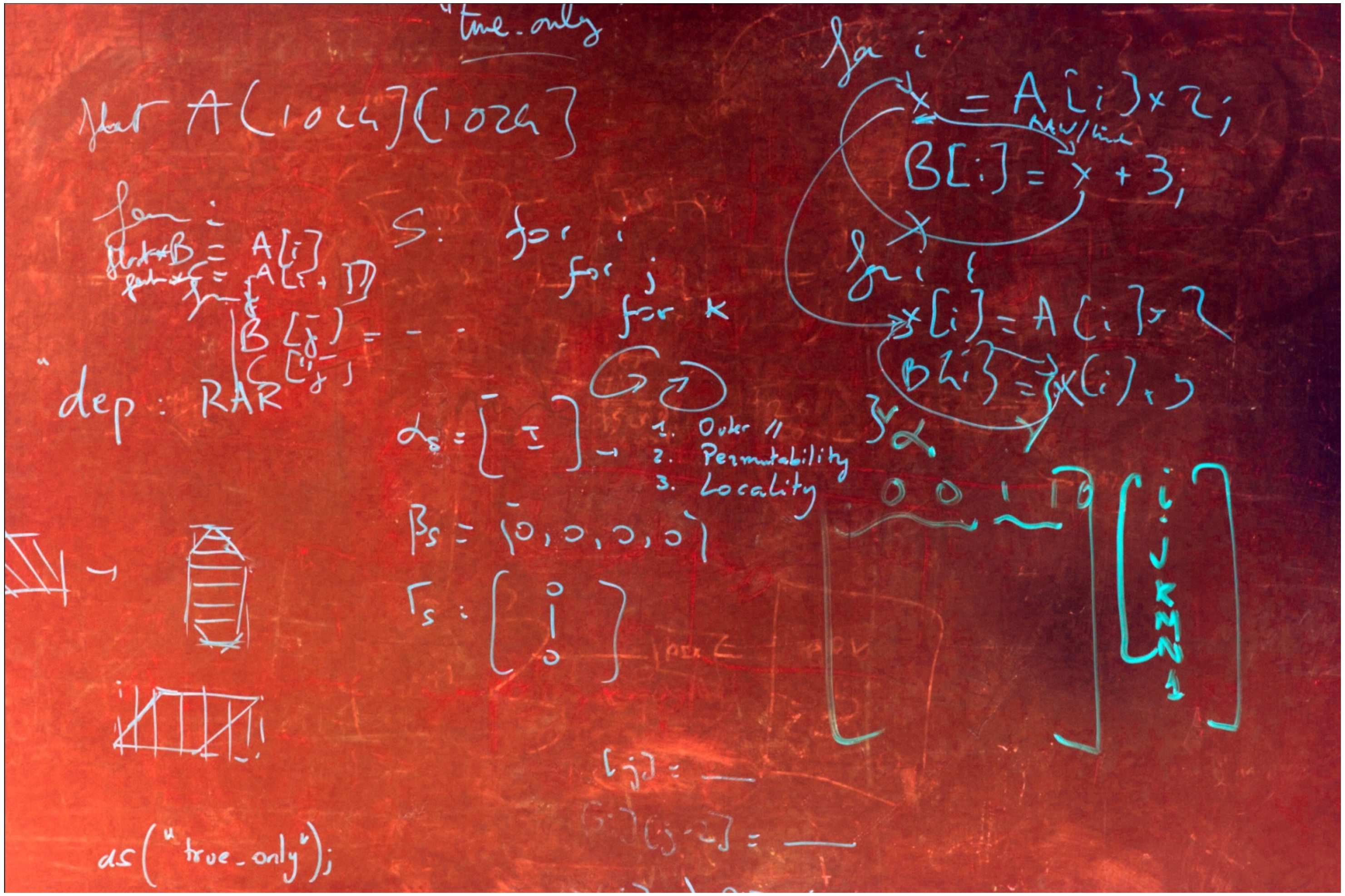
Hardware, tool, and algorithm facets in next generation low power HPEC (High Performance Embedded Computing) that should shape benchmarking and evaluation strategy



Disclaimer

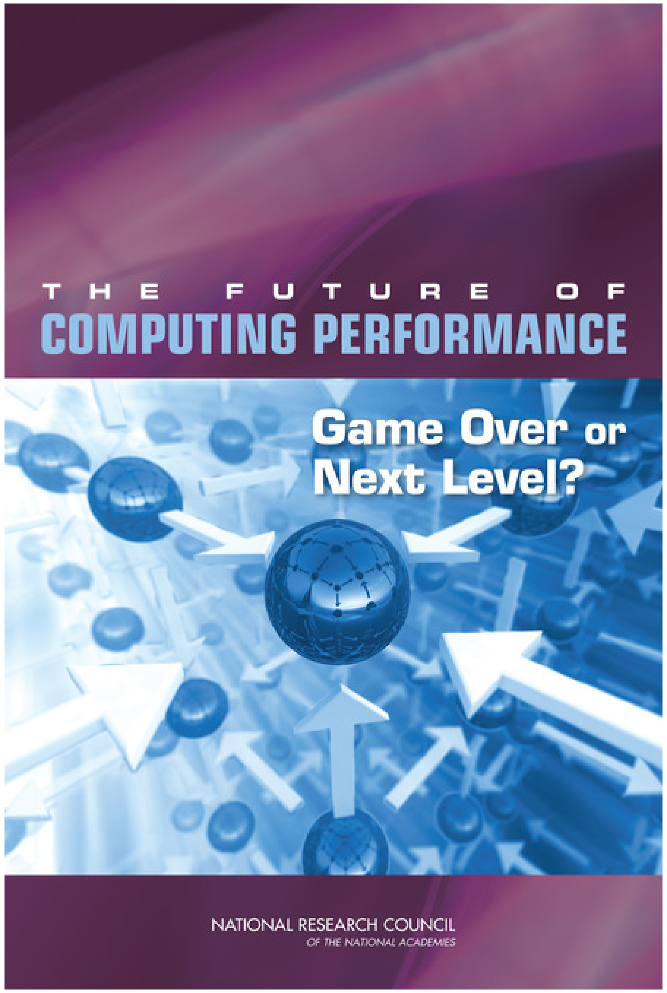
These are research projects, not all in Reservoir products.

No warranties.

Patent pending technologies.

Game Over or Next Level?

"A high-level performance-portable programming model is the only way to restart the virtuous cycle..."



Outline

Hardware Facets

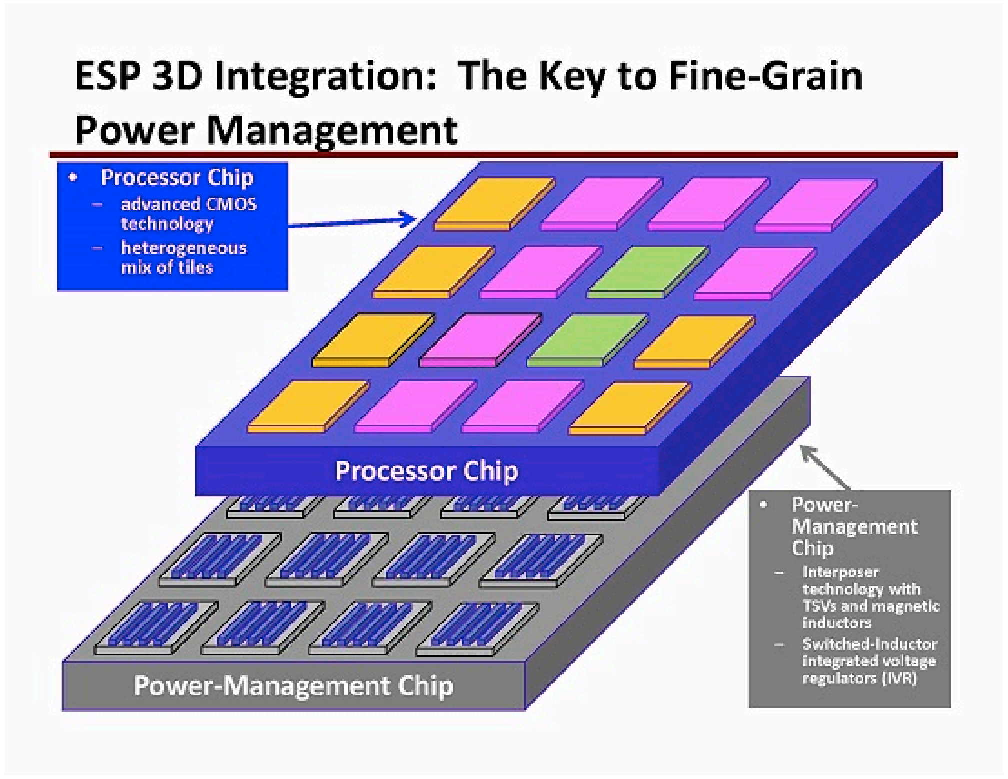
Tool Facets

Algorithm Facets

Benchmarking and Evaluation Implications

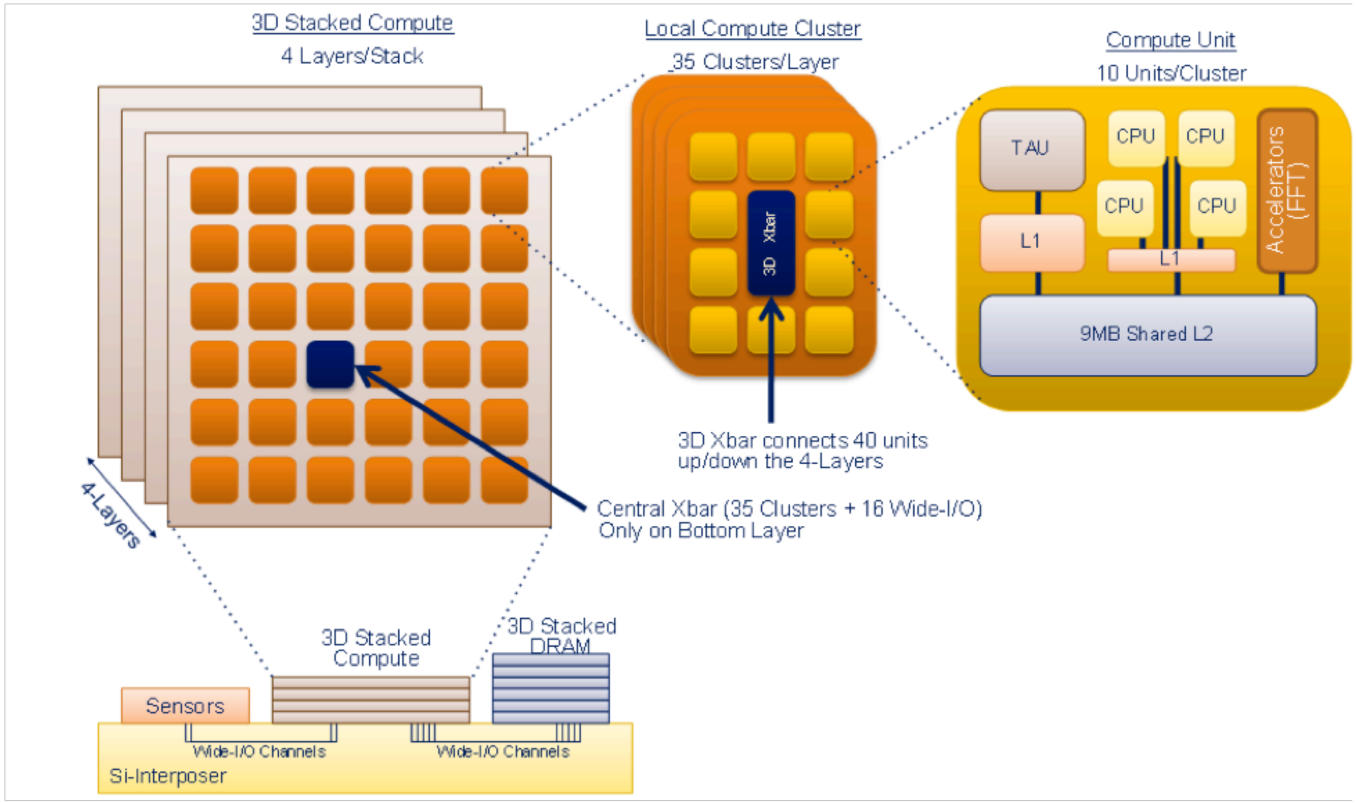
Hardware Facets

Nimble, Fine Grained Per Core Power Controls


ESP 3D Integration: The Key to Fine-Grain Power Management
• Processor Chip
– advanced CMOS technology
– heterogeneous mix of tiles
Processor Chip
Power-Management Chip
• Power-Management Chip
– Interposer technology with TSVs and magnetic inductors
– Switched-Inductor integrated voltage regulators (IVR)

"Embedded Scalable Processor (ESP)"
Columbia University
Luca Carloni (PI), 2014

More Cores, Deeper Hierarchy, NTV

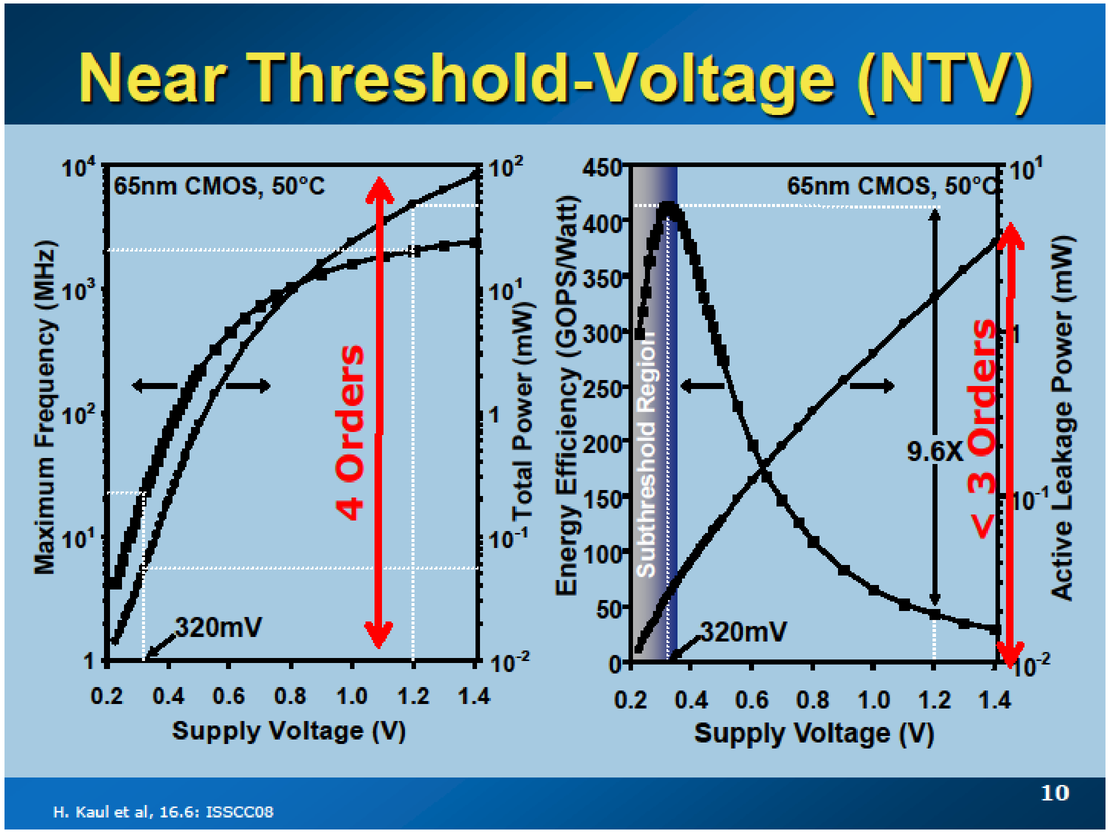


"Energy Efficient 3D Near-Threshold Computing Systems for Future Embedded Applications"
Trevor Mudge
University of Michigan, 2014

NTV Operation

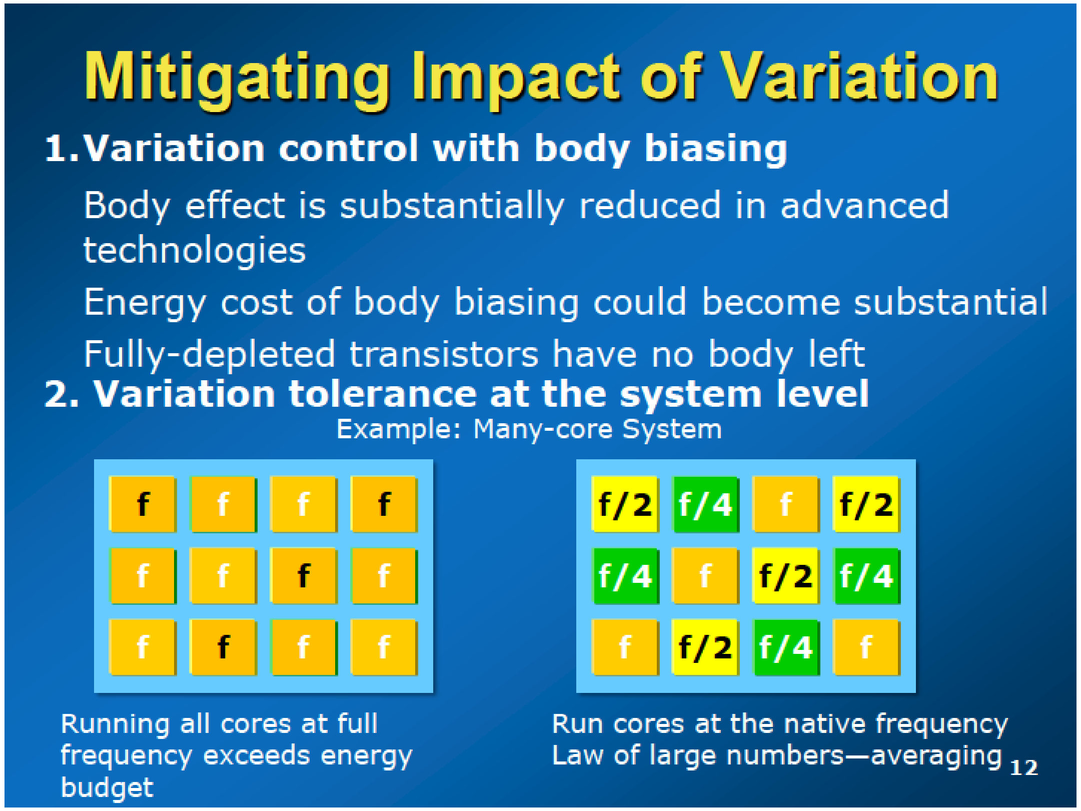
NTV Benefit:
4X Increase in Power Efficiency

NTV Cost:
Need more CONCURENCY
Greater leakage variation
Greater f_max variation
Bigger relative register files



Shekhar Borkar
IPDPS keynote
2013

NTV ➔ Cores running at varied clocks



Mitigating Impact of Variation

1. Variation control with body biasing
   - Body effect is substantially reduced in advanced technologies
   - Energy cost of body biasing could become substantial
   - Fully-depleted transistors have no body left
2. Variation tolerance at the system level

Example: Many-core System

f	f	f	f
f	f	f	f
f	f	f	f

Running all cores at full frequency exceeds energy budget

f/2	f/4	f	f/2
f/4	f	f/2	f/4
f	f/2	f/4	f

Run cores at the native frequency
Law of large numbers—averaging 12

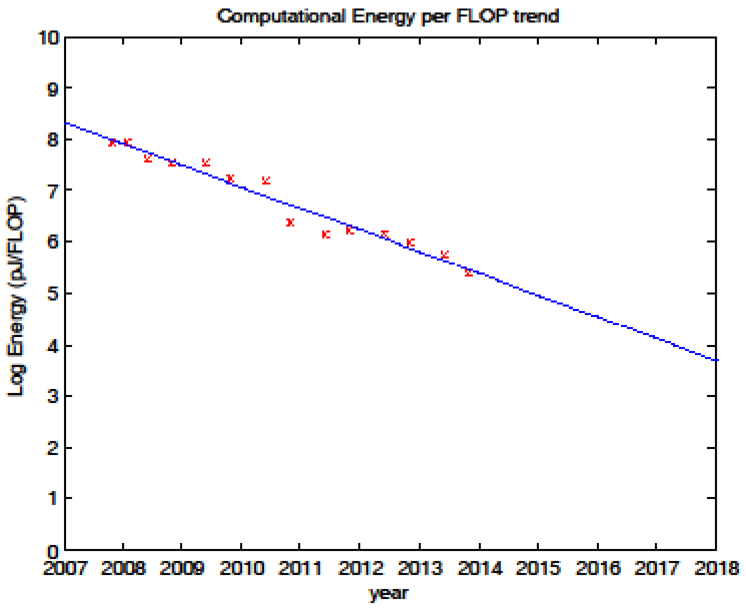
Cost:

More complex mapping and LOAD BALANCING

Shekhar Borkar
IPDPS keynote
2013

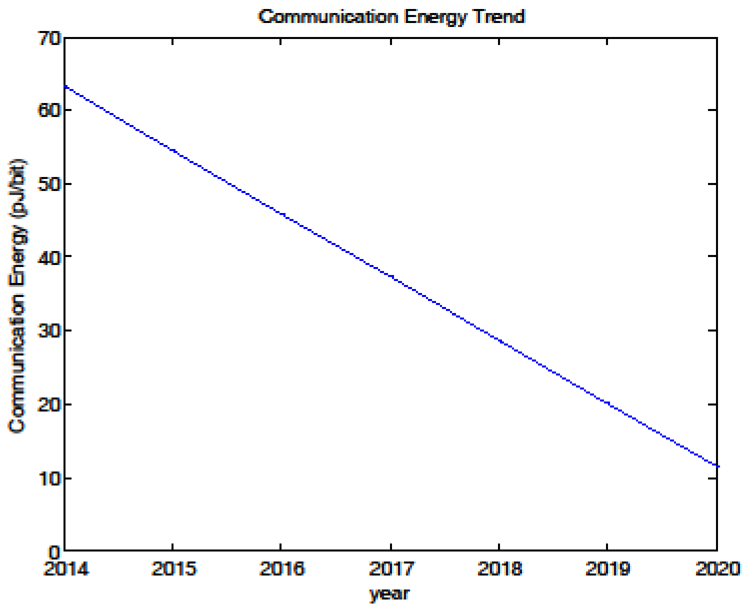
Energy Cost Trends: Arithmetic vs. Communications

Arithmetic



Log Scale (improving exponentially; i.e., Moore's Law)

Communications



Linear Scale (improving only linearly!)

Needed:
Communication-Avoiding Compilation,
Communication-Avoiding Algorithms

Other Hardware Trends

Heterogeneity

Compute Everywhere (in Network, near Memory, ...)

Sea of Tiles

Specialization / Dark Silicon

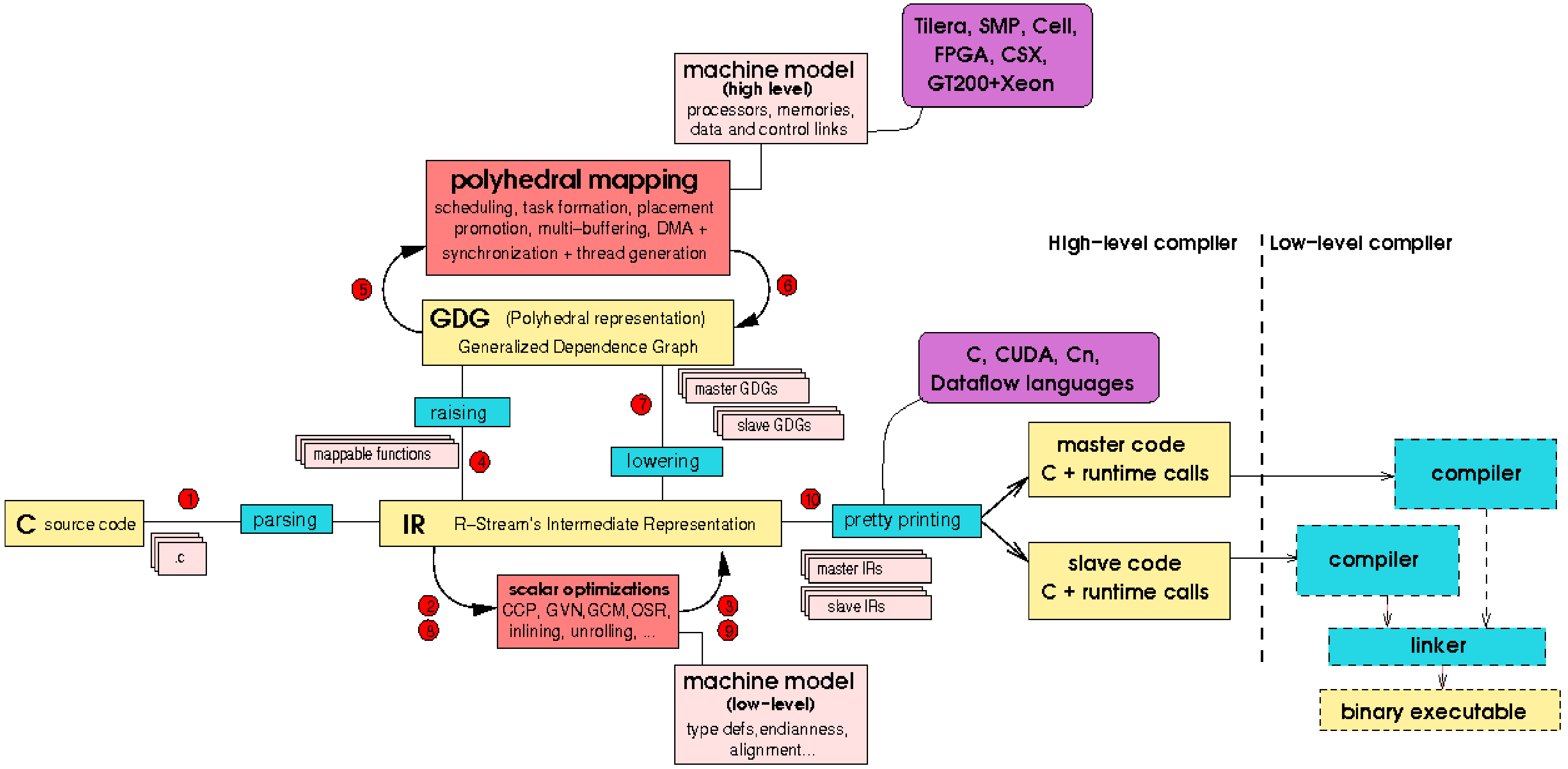
Explicit Communication (DMA, rDMA, ...)

Explicit Scratchpad hierarchy management

...

Tool Facets

R-Stream : More CONCURRENCY

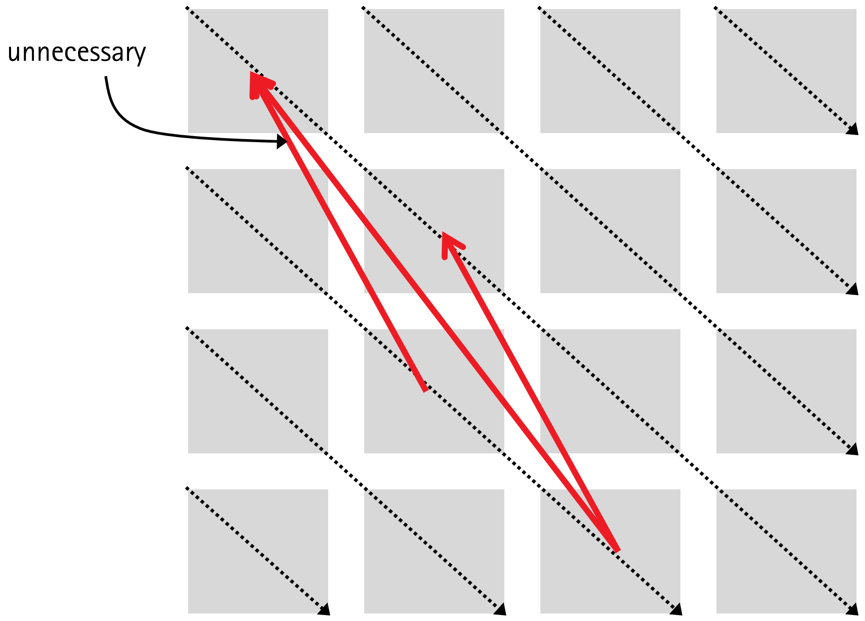

Tilera, SMP, Cell, FPGA, CSX, GT200+Xeon
machine model (high level) processors, memories, data and control links
polyhedral mapping
scheduling, task formation, placement promotion, multi-buffering, DMA + synchronization + thread generation
5
6
GDG (Polyhedral representation)
Generalized Dependence Graph
master GDGs
slave GDGs
raising
7
lowering
mappable functions
4
High-level compiler
Low-level compiler
C, CUDA, Cn, Dataflow languages
master code C + runtime calls
compiler
C source code
1
parsing
.c
IR R-Stream's Intermediate Representation
10
pretty printing
master IRs
slave IRs
slave code C + runtime calls
compiler
2
8
scalar optimizations CCP, GVN, GCM, OSR, inlining, unrolling, ...
3
9
linker
binary executable
machine model (low-level) type defs, endianness, alignment...

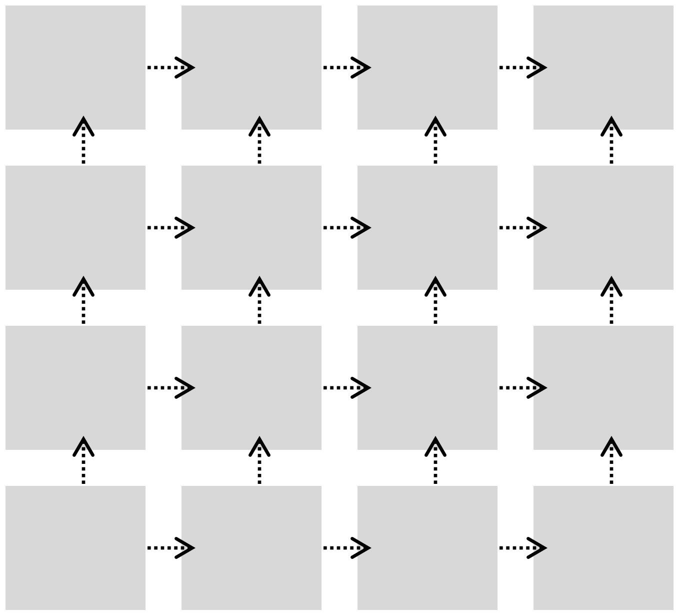
Targeting Task Graph Execution Models

Move beyond barrier synchronization

- Expose additional CONCURRENCY
- Increase opportunities for LOAD BALANCING
- Support runtimes that scale to 1000's of cores

Example: Givens QR mapping only requires neighbor syncs

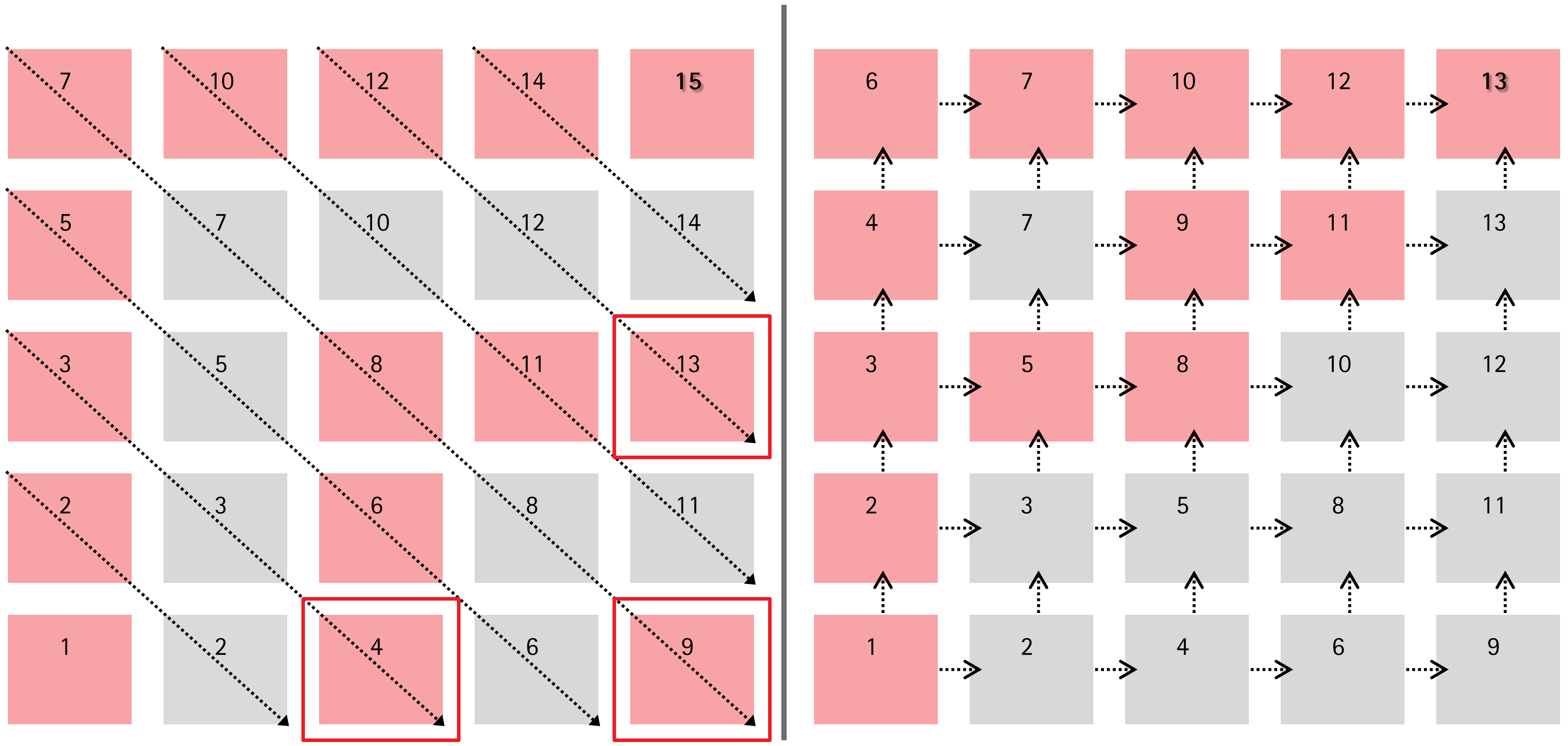




Benefits

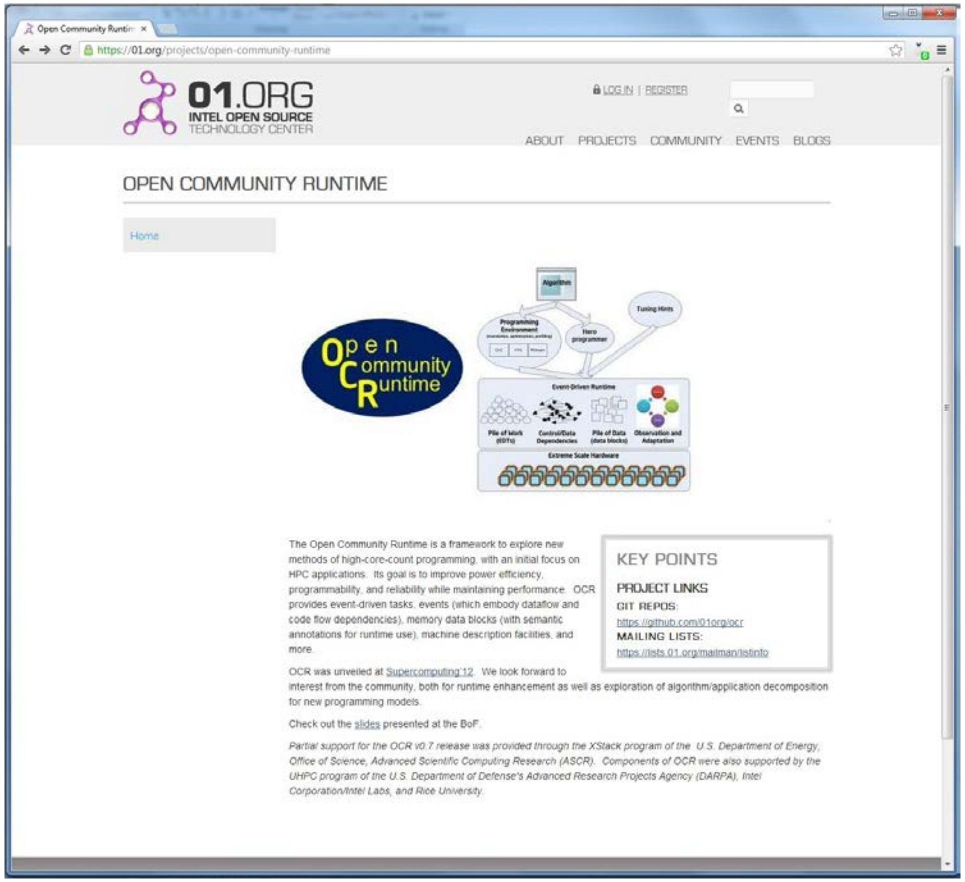
Better efficiency, shorter latency, lower power usage

Dynamic optimizations (locality, power, energy proportionality)



2 threads, 25 tasks

Open Community Runtime



OPEN COMMUNITY RUNTIME

The Open Community Runtime is a framework to explore new methods of high-core-count programming, with an initial focus on HPC applications. Its goal is to improve power efficiency, programmability, and reliability while maintaining performance. OCR provides event-driven tasks, events (which embody dataflow and code flow dependencies), memory data blocks (with semantic annotations for runtime use), machine description facilities, and more.

KEY POINTS

PROJECT LINKS

GIT REPOS:
https://github.com/01org/ocr

MAILING LISTS:
https://lists.01.org/mailman/listinfo

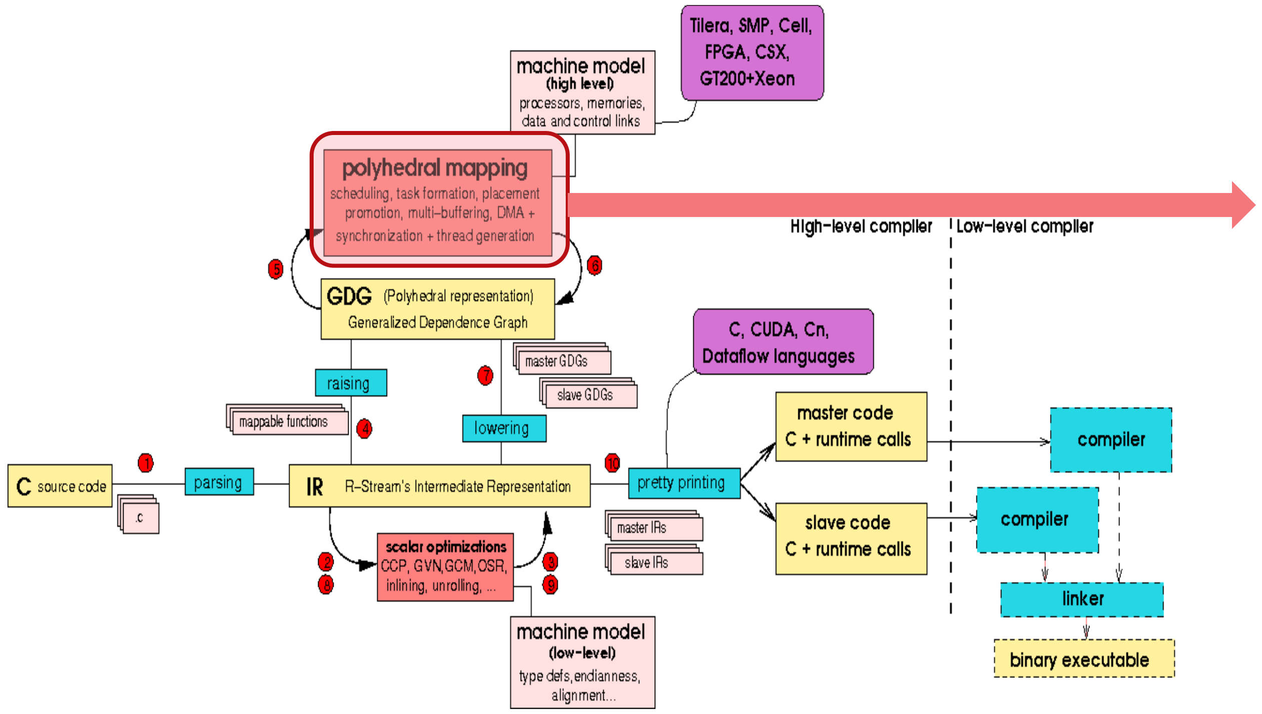
OCR was unveiled at Supercomputing'12. We look forward to interest from the community, both for runtime enhancement as well as exploration of algorithm/application decomposition for new programming models.

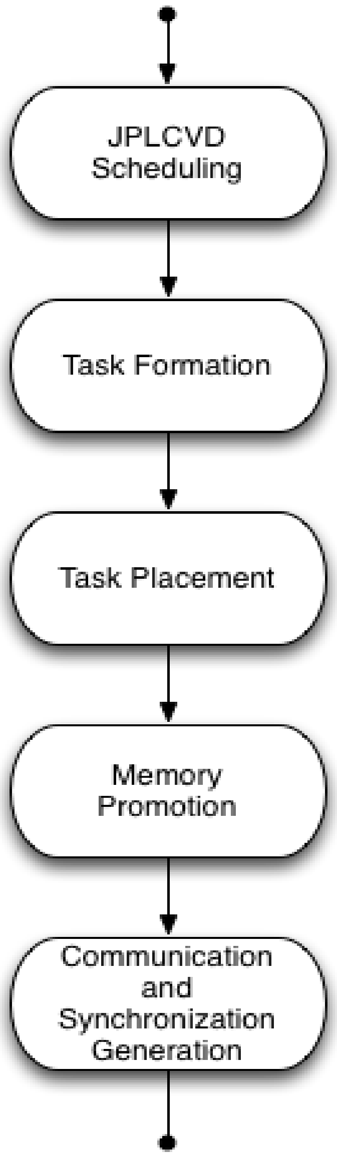
Check out the slides presented at the BoF.

Partial support for the OCR v0.7 release was provided through the XStack program of the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR). Components of OCR were also supported by the UHPC program of the U.S. Department of Defense's Advanced Research Projects Agency (DARPA), Intel Corporation/Intel Labs, and Rice University.

https://01.org/projects/open-community-runtime

Polyhedral Mapper in R-Stream


Tilera, SMP, Cell, FPGA, CSX, GT200+Xeon
machine model (high level) processors, memories, data and control links
polyhedral mapping
scheduling, task formation, placement promotion, multi-buffering, DMA + synchronization + thread generation
High-level compiler
Low-level compiler
GDG (Polyhedral representation) Generalized Dependence Graph
master GDGs
slave GDGs
raising
lowering
mappable functions
C, CUDA, Cn, Dataflow languages
master code C + runtime calls
slave code C + runtime calls
compiler
compiler
linker
binary executable
C source code
.c
parsing
IR R-Stream's Intermediate Representation
pretty printing
master IRs
slave IRs
scalar optimizations CCP, GVN,GCM,OSR, inlining, unrolling, ...
machine model (low-level) type defs, endianness, alignment...
1
2
3
4
5
6
7
8
9
10



Communication-Avoiding Scheduling

Affine Scheduling – JPLCVD

- Jointly optimize parallelism, locality, contiguity, vectorization and data layout

Local Memory Buffer Management

- Scratchpad and/or "Virtual Scratchpad"

Minimize copying and memory traffic in local memory

- "Right size" local buffers and round-robin reuse

Communication-Avoiding Compilation

```
for k=1,n {
    for j=1,n {
       for i=1,n {
          c[3*(i-1)+1] = A[k][j][i+1];
          c[3*(i-1)+2] = A[k][j][i+2];
          c[3*(i-1)+3] = A[k][j][i+3];
             ...
          X[k][j][i] = c[3*(i-1)+1] *
                       c[3*(i-1)+2] +
                       c[3*(i-1)+3];
       }
    }
 }
```

Original Code (Covariance Calculation from STAP)

Communication-Avoiding Compilation

```
for k=1,n {
   for j=1,n {
       for i=1,n {
          if (i == 1) {
             c[3*(i-1)+1] = A[k][j][i+1];
             c[3*(i-1)+2] = A[k][j][i+2];  // S1
             c[3*(i-1)+3] = A[k][j][i+3];  // S2
               ...
             X[k][j][i] = c[3*(i-1)+1] *
                          c[3*(i-1)+2] +
                          c[3*(i-1)+3];
          }
          if (i == 2) {
             c[3*(i-1)+1] = A[k][j][i+1];  // S3
             c[3*(i-1)+2] = A[k][j][i+2];  // S4
             c[3*(i-1)+3] = A[k][j][i+3];  // S5
                ...
             X[k][j][i] = c[3*(i-1)+1] *
                          c[3*(i-1)+2] +
                          c[3*(i-1)+3];   // S6
          }
          if (i > 2) {
             ...
          }
       }
    }
}
```

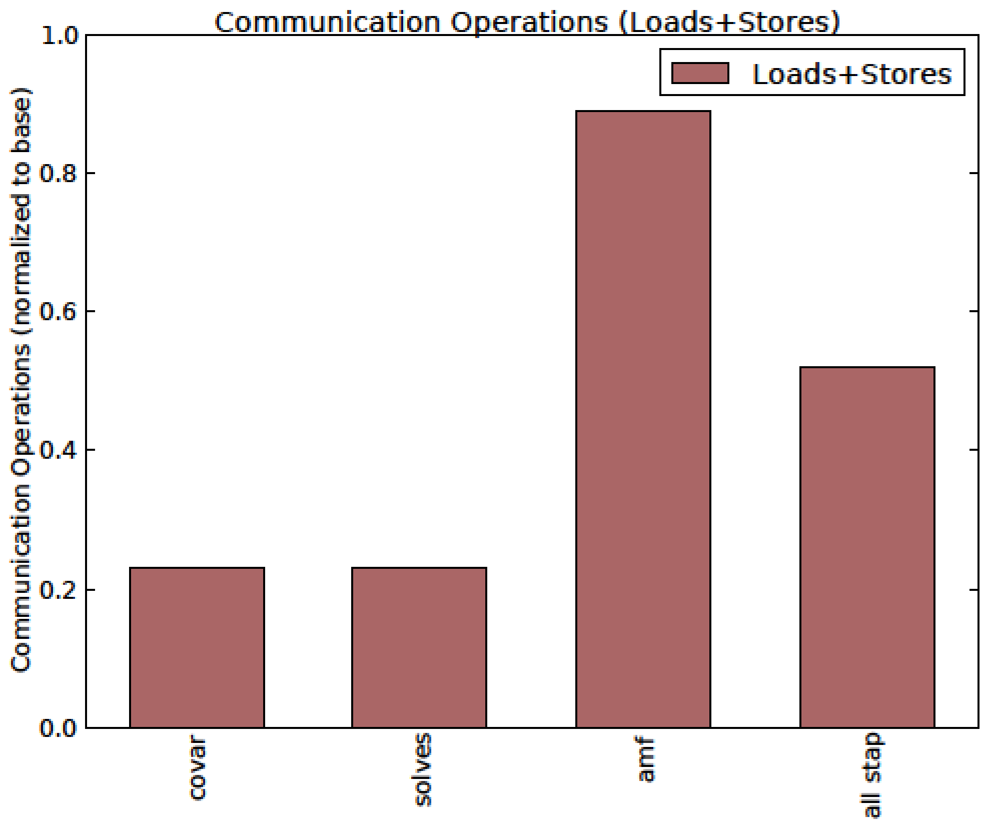
Code with two iterations of i loop peeled for illustration

Communication-Avoiding Compilation

```
for k=1,n {
   for j=1,n {
      for i=1,n {
         if (i == 1) {
            c[3*(i-1)+1] = A[k][j][i+1];
            c[3*(i-1)+2] = A[k][j][i+2];  // S1
            c[3*(i-1)+3] = A[k][j][i+3];  // S2
               ...
            X[k][j][i] = c[3*(i-1)+1] *
                         c[3*(i-1)+2] +
                         c[3*(i-1)+3];
         }
         if (i == 2) {
            //c[3*(i-1)+1] = A[k][j][i+1];  // S3 : Removed - same as S1 read (S3->S1 RAR)
            //c[3*(i-1)+2] = A[k][j][i+2];  // S4 : Removed - same as S2 read (S4->S2 RAR)
            c[(i-1)\%3] = A[k][j][i+3];     // S5: Access fn change: c[3*(i-1)+3] -> c[((3*(i-1)+3)/3-1)\%3]
                ...
            X[k][j][i] = c[3*(i-2)+2] *  // S6: Changed access fn to point to element of c loaded in S1
                                         //   c[3*(i-1)+1] -> c[3*(i-2)+2] (S6->S3 RAW + S3->S1 RAR)
                         c[3*(i-2)+3] +  // S6: Changed access fn to point to element of c loaded in S2
                                         //   c[3*(i-1)+2] -> c[3*(i-2)+3] (S6->S4 RAW + S4->S2 RAR)
                         c[(i-1)\%3];    // S6: Access fn change: c[3*(i-1)+3] -> c[((3*(i-1)+3)/3-1)\%3]
         }
         if (i > 2) {
            ...
         }
      }
   }
}
```

Code with communication-avoiding optimizations applied

Communications-Savings (GPGPU – Kayla)

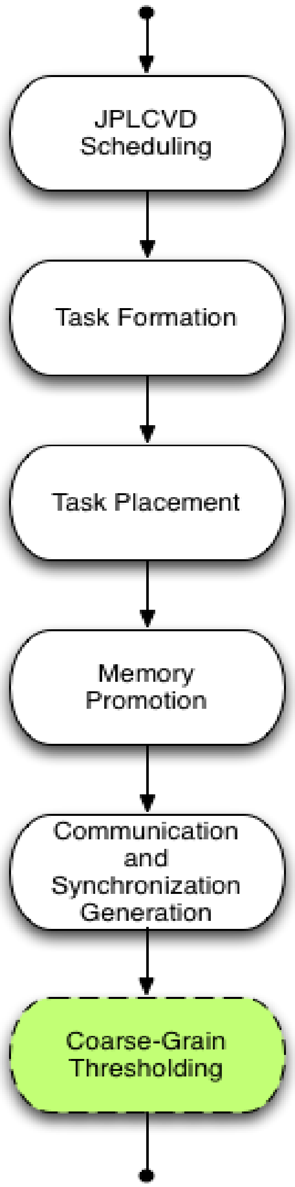


New R-Stream Optimizations Decrease Communications by 2X

Energy Proportional Scheduling in R-Stream

Coarse-grain Thresholding

- Generate lightweight runtime checks to apply appropriate level of energy optimization on key system components using the Power API

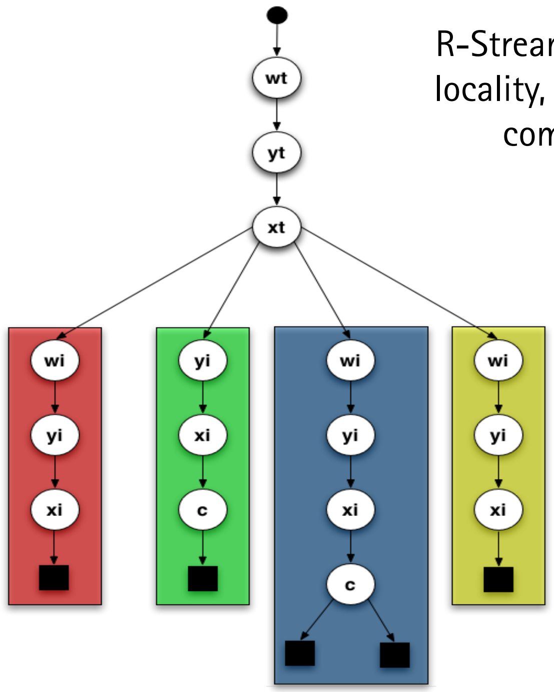

JPLCVD Scheduling
Task Formation
Task Placement
Memory Promotion
Communication and Synchronization Generation
Coarse-Grain Thresholding

EPS in R-Stream

```
for(w=0;w<W;w++)     // W=56
 for(y=0;y<Y0;y++)   // Y0=1021
  for(x=0;x<X0;x++)  // X0=1021
   for(c=0;c<C;c++)  // C=32
    out[w][y][x] +=
    in[y][x][c]*filt[w][0][0][c]+
    in[y+1][x][c]*filt[w[[1][0][c]+
        …
    in[y+2][x+2][c]*filt[w][2][2][c];
```

```
for(wt=0;wt<=6;wt++){
 for(yt=0;yt<=7;yt++){
  for(xt=0;xt<=7;xt++){
   for(wi=…){
     for(yi=0…){
      for(xi=0…){
       … /* load array out */
   }}}
   for(yi=…){
     for(xi=0…){
      for(c=0…){
       … /* load array in */
   }}}
   for(wi=0;wi<=7;wi++){
    for(yi=0;yi<=min(…,127);yi++){
     for(xi=0;xi<=min(…,127);xi++){
      for(c=0;c<=31;c++){
       … /* load array filt */
       out_l[8*wt+wi][…][…] += …
   }}}}
   for(wi=…){
     for(yi=0…){
      for(xi=0…){
       … /* store array out */
   }}}
```

R-Stream applies parallelism, locality, tiling [8,128,128,32], communication gen



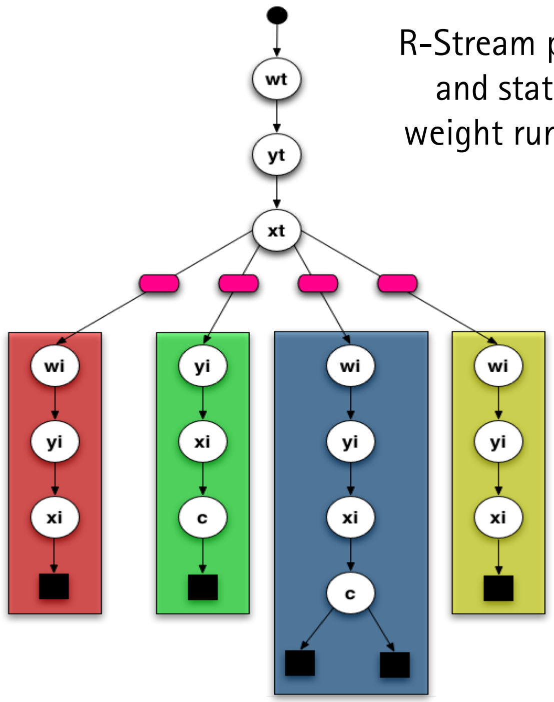
Betatree representation in R-Stream

EPS in R-Stream

```
for(w=0;w<W;w++)     // W=56
 for(y=0;y<Y0;y++)   // Y0=1021
  for(x=0;x<X0;x++)  // X0=1021
   for(c=0;c<C;c++)  // C=32
    out[w][y][x] +=
    in[y][x][c]*filt[w][0][0][c]+
    in[y+1][x][c]*filt[w[[1][0][c]+
        …
    in[y+2][x+2][c]*filt[w][2][2][c];
```

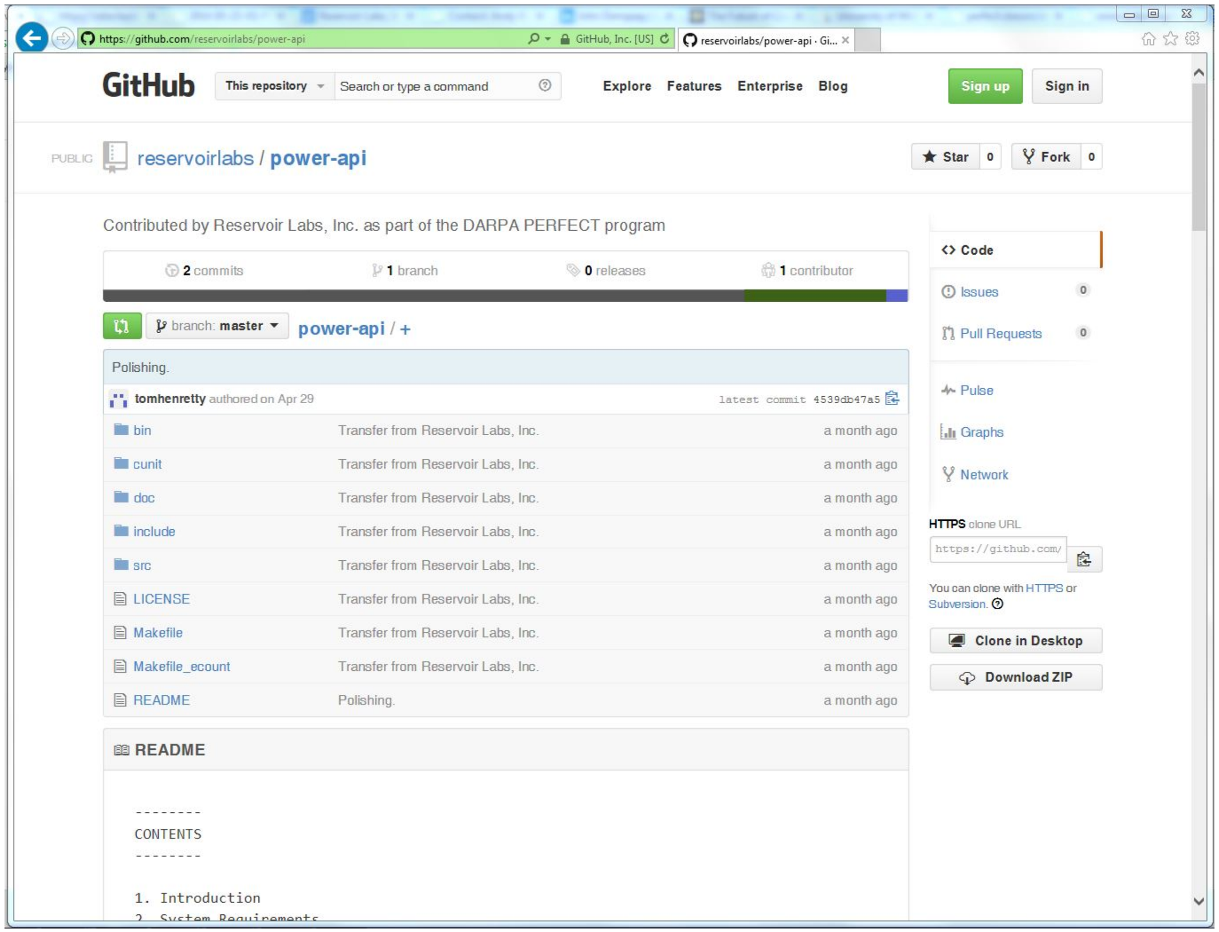
```
for(wt=0;wt<=6;wt++){
 for(yt=0;yt<=7;yt++){
  for(xt=0;xt<=7;xt++){
   R-Stream Power API call();
   for(wi=…){
     for(yi=0…){
      for(xi=0…){
       … /* load array out */ }}}
   R-Stream Power API call();
   for(yi=…){
     for(xi=0…){
      for(c=0…){
       … /* load array in */ }}}
   R-Stream Power API call();
   for(wi=0;wi<=7;wi++){
    for(yi=0;yi<=min(…,127);yi++){
     for(xi=0;xi<=min(…,127);xi++){
      for(c=0;c<=31;c++){
       … /* load array filt */
       out_l[8*wt+wi][…][…] += …
   }}}}
   R-Stream Power API call();
   for(wi=…){
     for(yi=0…){
      for(xi=0…){
       … /* store array out */ }}}
```

R-Stream performs EPS strategy and statically inserts light-weight runtime power API calls



Betatree representation in R-Stream

Power API



https://github.com/reservoirlabs/power-api

GitHub | This repository | Search or type a command | Explore | Features | Enterprise | Blog | Sign up | Sign in

PUBLIC reservoirlabs / power-api | Star 0 | Fork 0

Contributed by Reservoir Labs, Inc. as part of the DARPA PERFECT program

2 commits | 1 branch | 0 releases | 1 contributor

branch: master | power-api / +

Polishing.

tomhenretty authored on Apr 29 — latest commit 4539db47a5

bin	Transfer from Reservoir Labs, Inc.	a month ago
cunit	Transfer from Reservoir Labs, Inc.	a month ago
doc	Transfer from Reservoir Labs, Inc.	a month ago
include	Transfer from Reservoir Labs, Inc.	a month ago
src	Transfer from Reservoir Labs, Inc.	a month ago
LICENSE	Transfer from Reservoir Labs, Inc.	a month ago
Makefile	Transfer from Reservoir Labs, Inc.	a month ago
Makefile_ecount	Transfer from Reservoir Labs, Inc.	a month ago
README	Polishing.	a month ago

README

```
--------
CONTENTS
--------

1. Introduction
```

https://github.com/reservoirlabs/power-api

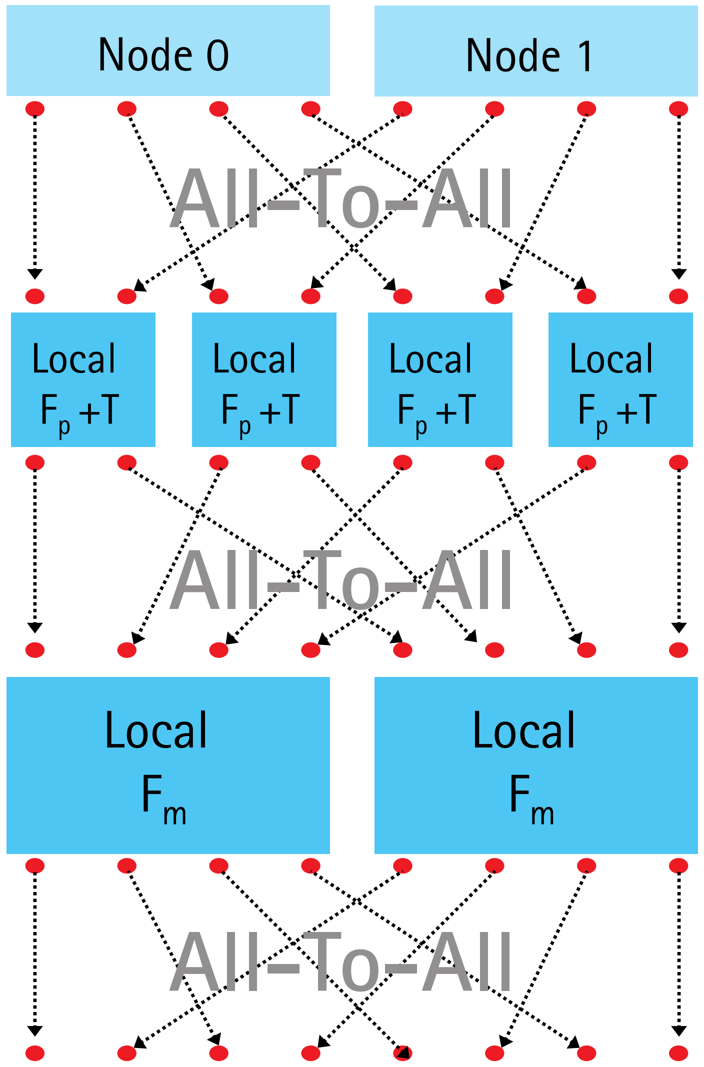
Algorithm Facets

Traditional Parallel FFT Approach in 1D

Assume N = M*P Points and Apply The Operator:

- $F_n = (I_p \otimes F_m)(F_p \otimes I_m)\Pi$
- Perform Global Bit Reversal
- Perform Local FFTs and Global Transpose
- Apply Twiddle Factors
- Perform Local FFTs and Global Bit Reversal

- REQUIRES 3 GLOBAL ALL-TO-ALL CALLS!



Alternative Low-Communication FFTs Approximate Computing

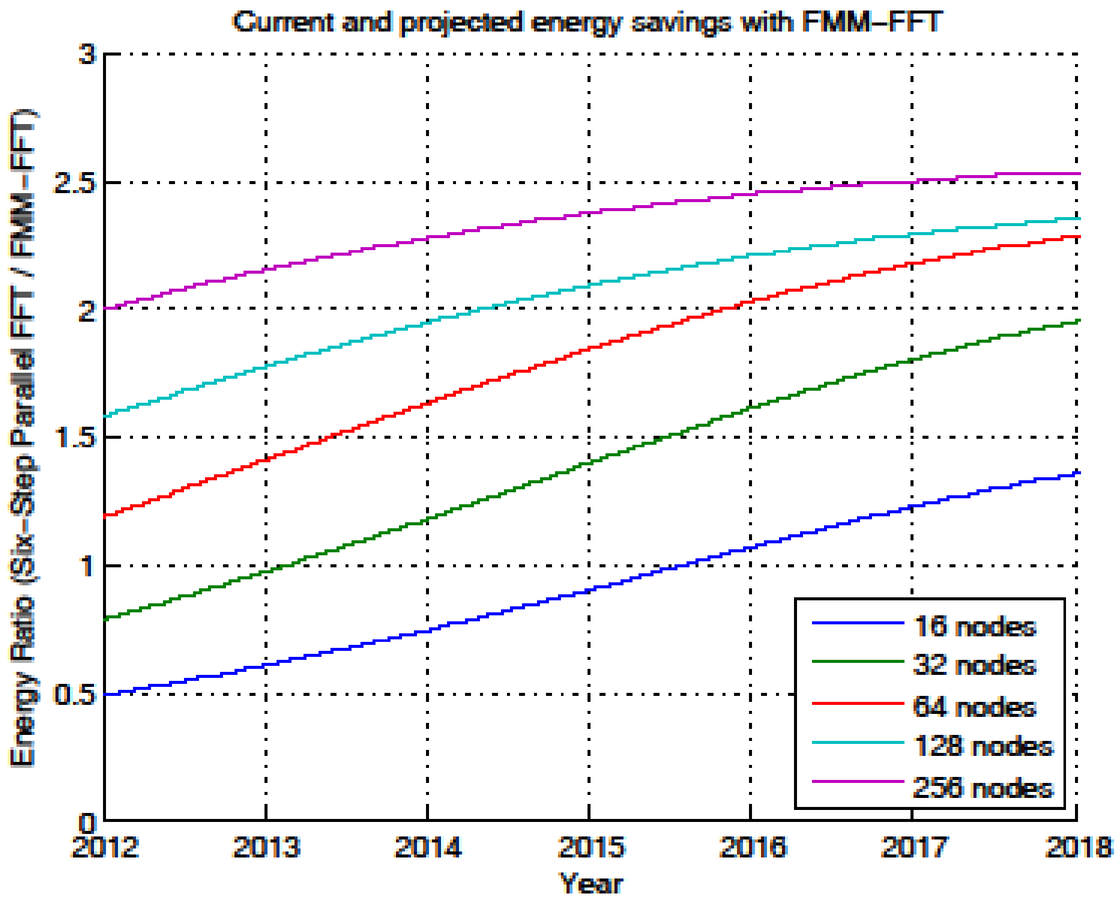
Reducing Global Communication from 3 to ~1

- Tang et. al (SC'12) use an oversampling approach while mentioning older approach:
- Edelman et. al. (SIAM J.SciComp'97) has largely been ignored due to previous lack of interest and reliance on FMMs:
- Refactor the operator as $F_n = (I_p \otimes F_m)(F_p \otimes I_m)M\Pi$ with factor matrices $M = diag(I_m, C^1, ..., C^{p-1})$
- *C* matrices applied with FMM to reduce global communication while arithmetic cost increases. Results (N=10^7 points on Intel(R) Xeon(R) CPU X5650 @ 2.67GHz CPUs and QDR InfiniBand):

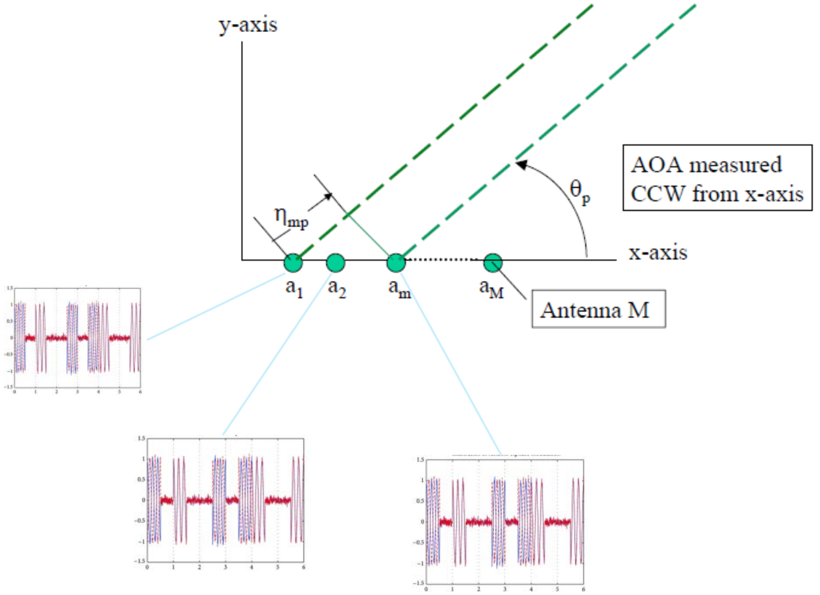
NP	T^{ops}_{FMM}	$T^{ops}_{FFT_{FMM}}$	C^{msgs}_{FMM}	C^{msgs}_{A2A}	C^{msgs}_{TOT}	$W^{FFT_P}_{tot}(s)$	$W^{FFT_{FMM}}_{tot}(s)$
2	$4.92e+09$	$5.92e+09$	$8.58e+02$	$4.19e+06$	$4.20e+06$	$1.44e+01$	$3.50e+00$
4	$3.33e+09$	$3.83e+09$	$2.39e+03$	$3.15e+06$	$3.15e+06$	$3.83e+00$	$2.58e+00$
16	$9.94e+08$	$1.12e+09$	$1.05e+04$	$9.83e+05$	$9.94e+05$	$5.74e-01$	$9.54e-01$
32	$5.11e+08$	$5.74e+08$	$2.02e+04$	$5.08e+05$	$5.28e+05$	$2.23e-01$	$4.86e-01$
64	$2.60e+08$	$2.91e+08$	$3.81e+04$	$2.58e+05$	$2.96e+05$	$1.45e-01$	$2.96e-01$
128	$1.33e+08$	$1.48e+08$	$7.07e+04$	$1.30e+05$	$2.01e+05$	$1.14e-01$	$2.48e-01$

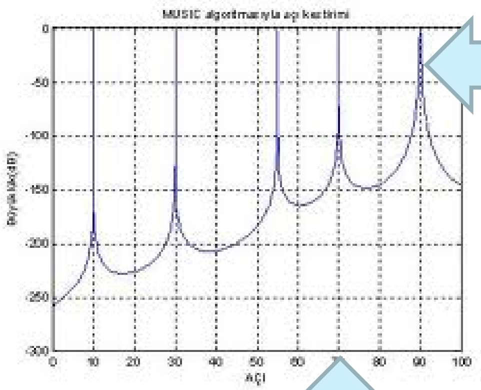
- Despite ~10x increase in arithmetic, ~3x reduction in communication results in only 2x slower runtimes

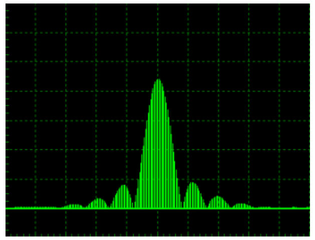
Energy Savings for FMM-FFT



Angle of Arrival/Spectrum Sensing Problem







Determine spectrum of each emitter

Locate emitters in Angle

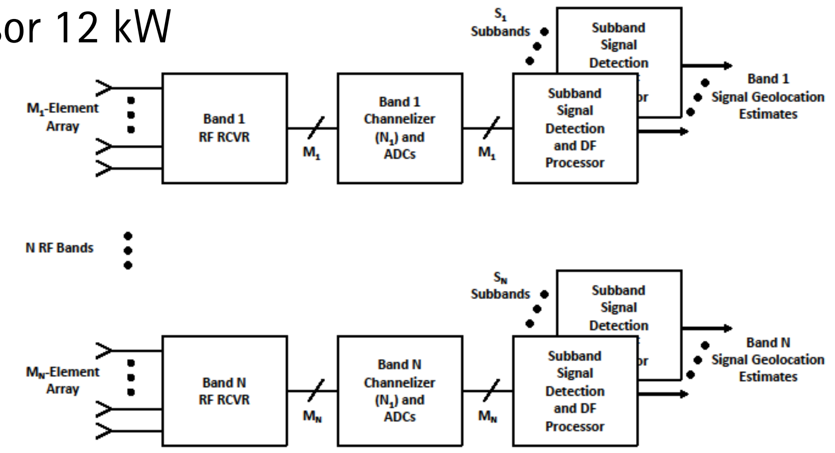
Radar Band	Frequency (GHz)	Wavelength (cm)
Millimeter	40 to 100	0.75 to 0.30
Ka	26.5 to 40	1.1 to 0.75
K	18 to 26.5	1.7 to 1.1
Ku	12.5 to 18	2.4 to 1.7
X	8 to 12.5	3.75 to 2.4
C	4 to 8	7.5 to 3.75
S	2 to 4	15 to 7.5
L	1 to 2	30 to 15
UHF	0.3 to 1	100 to 30

Fastest ADC can only sample a fraction of radar band

Compressive Sensing - kW to W

Processor 12 kW

Sensor 12 kW

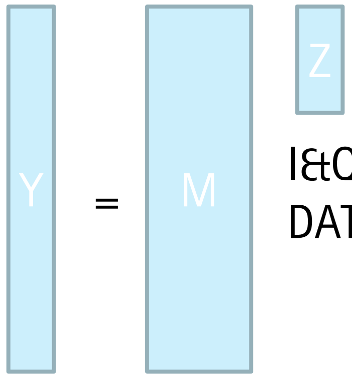


S_1 Subbands
Subband Signal Detection
Band 1 Signal Geolocation Estimates
M_1-Element Array
Band 1 RF RCVR
M_1
Band 1 Channelizer (N_1) and ADCs
M_1
Subband Signal Detection and DF Processor
N RF Bands
S_N Subbands
Subband Signal Detection
Band N Signal Geolocation Estimates
M_N-Element Array
Band N RF RCVR
M_N
Band N Channelizer (N_1) and ADCs
M_N
Subband Signal Detection and DF Processor

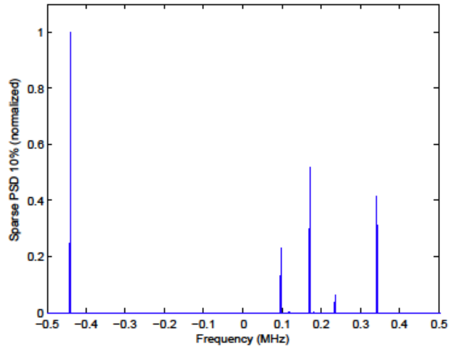
Classical Baseline

Sparse Image

DFT-like matrix



I&Q DATA



Sensor 100 W

Processor 100 W

Compressive Sensing

Benchmarking and Evaluation Implications

Hardware programming complexity increasing to save power; suites coded to the metal not portable

Many aspects of new programming complexity can be addressed by automatic tools – consider new tools in suites

- Provides some performance portability
- Input code for benchmarks should be "textbook form"

Algorithm landscape is changing too – expand suites

- Communication-avoiding algorithms
- Sparsity-exploiting algorithms (compressed sensing, etc.)
- Randomized algorithms
- Spectral support solvers

Need to consider system power