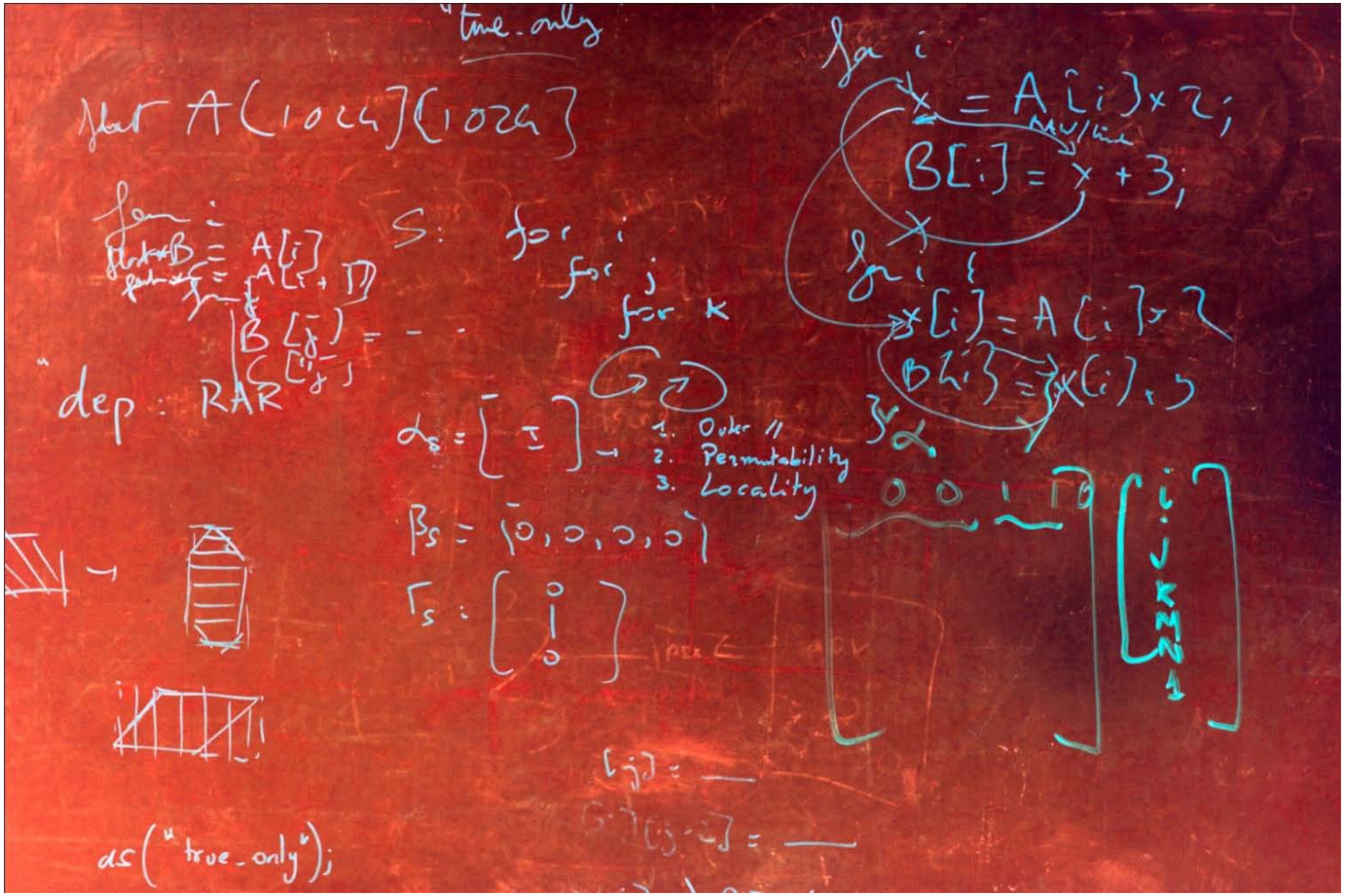
Automatic Parallelization and Locality Optimization of Beamforming Algorithms

Albert Hartono, Nicolas Vasilache, Cedric Bastoul, Allen Leung, Benoit Meister, Richard Lethin, and Peter Vouras[1]

Reservoir Labs, Inc.
[1]Naval Research Laboratory



Acknowledgment

Sponsors:

- Missile Defense Agency (Contract # W9113M-08-C-0146)
- DARPA
- DoE/DoD – other

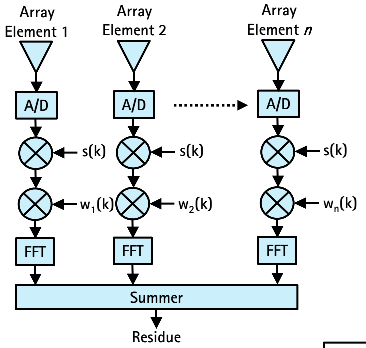
Subcontractor:

- Lockheed Martin MS2
  - Rick Pancoast
  - Scott Sawyer

The Problem

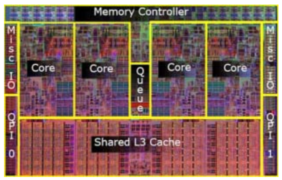
Increasingly complex application design

- Adaptive algorithms
- More elements
- Higher dimensionality
- Fused consideration (e.g., detection/imaging)

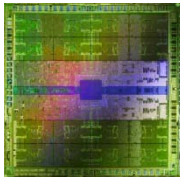


Increasingly complex hardware

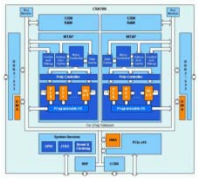
- More coarse-grained parallelism
- SIMD
- Data locality
- Proprietary programming models and languages (e.g., CUDA)
- Explicit resource management (memories, communication)



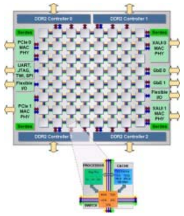
Intel Nehalem



Nvidia GPU



ClearSpeed CSX700

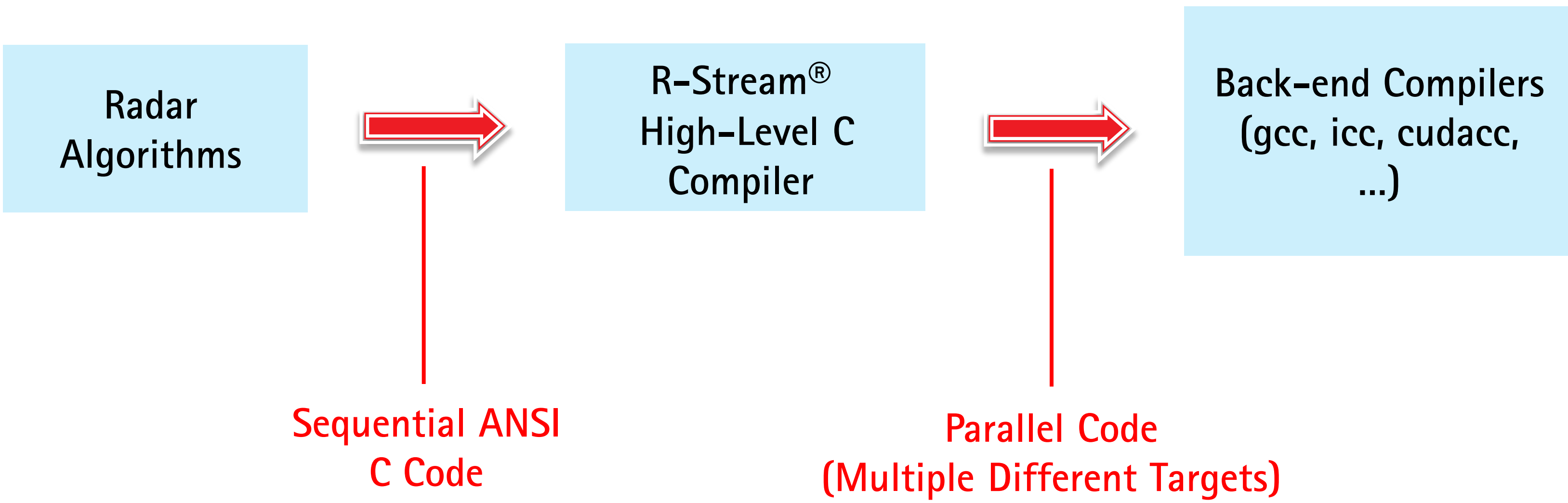


Tilera TILE64

Software challenges

- Open API programming language
- Performance
- Productivity
- Portability

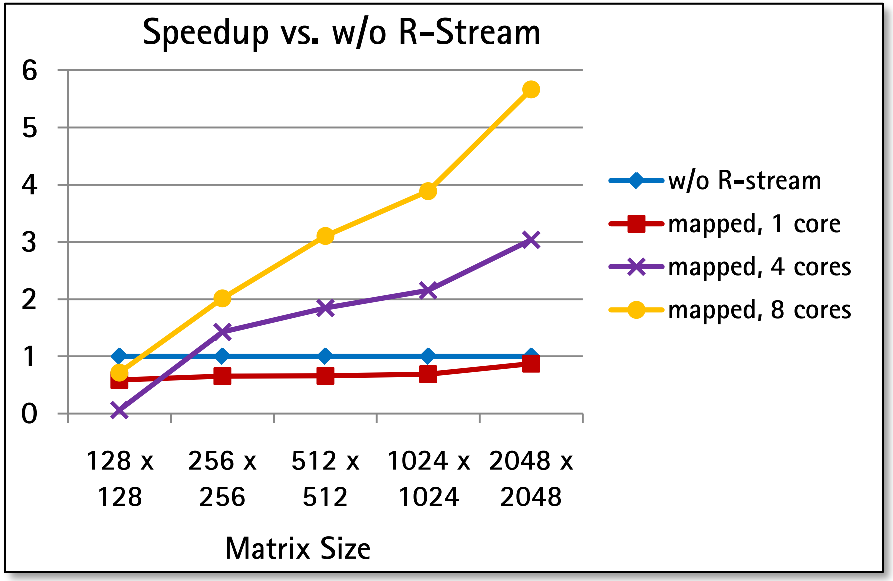
Simple Programming Flow

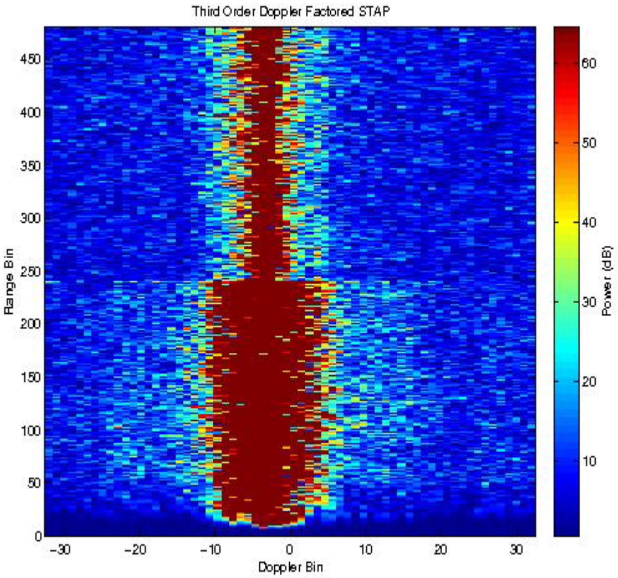


R-Stream® is an advanced high level compiler developed by Reservoir Labs, Inc.

Previously

Examined speedups for Givens QR decomposition algorithm as central component of an advanced STAP filter (Mitre RT-STAP benchmark)





RT-STAP Hard Profile (8 Processors)

But:

- QR is already in library. Where is benefit of auto parallelization?
- Practical radar algorithms are moving to incremental formulation (with lots of elements)
- QR is in the context of other parts of radar (weight application, etc)

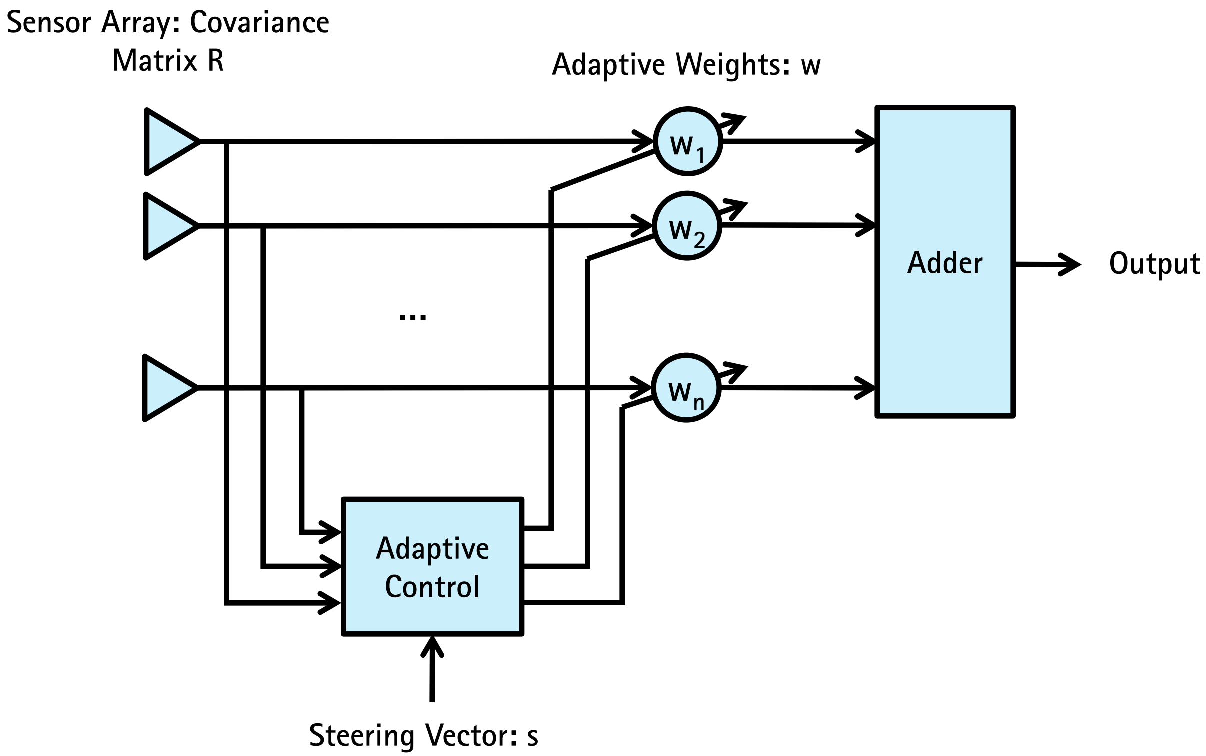
The Experiment

Examine automatic mapping of radar algorithms to advanced multi-core hardware

Key points:

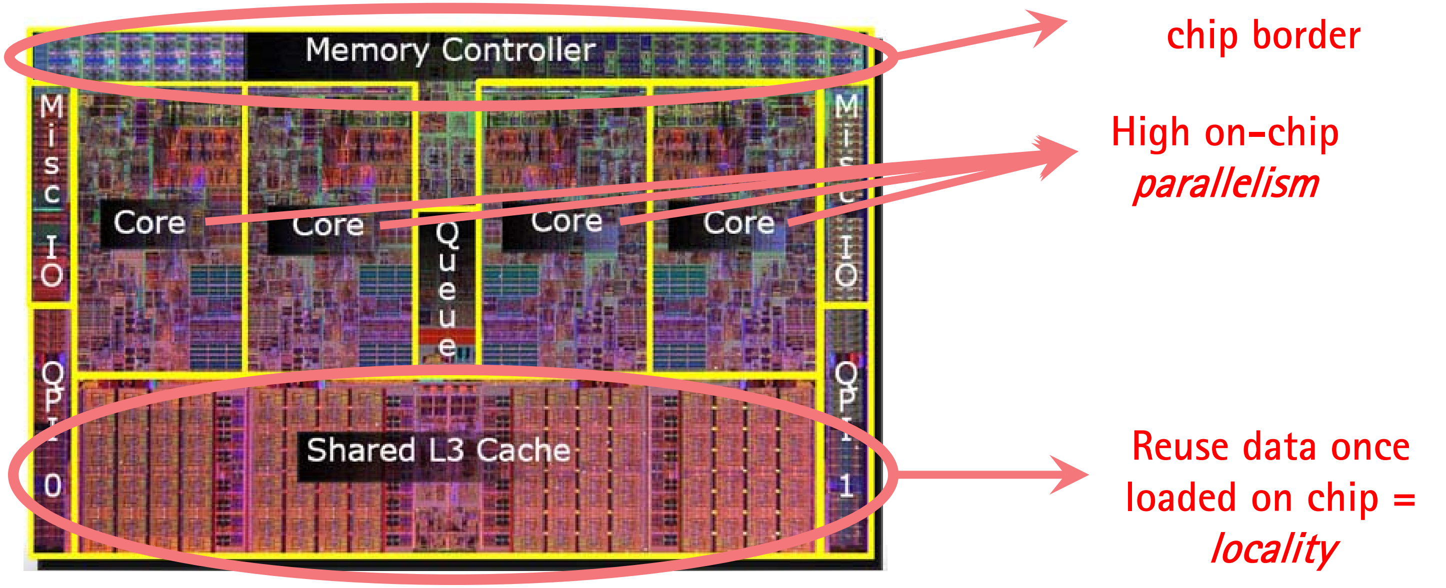
- Can we automatically optimize and show parallel speedups? What about locality?
- Can global optimization explore greater opportunities for speedups?
- How is the performance compared to libraries?

Adaptive Beamforming



Tradeoffs Between Parallelism and Locality

- Significant parallelism is needed to fully utilize all resources
- Locality is also critical to minimize communication
- Parallelism can come at the expense of locality



Our approach: R-Stream compiler exposes parallelism via *affine scheduling* that simultaneously augments locality using *loop fusion*

Parallelism/Locality Tradeoff Example

Array z gets expanded, to introduce another level of parallelism

Maximum distribution destroys locality

```
/*
 * Original code:
 *   Simplified CSLC-LMS
 */
for (k=0; k<400; k++) {
 for (i=0; i<3997; i++) {
  z[i]=0;
  for (j=0; j<4000; j++)
   z[i]= z[i]+B[i][j]*x[k][j];
 }
 for (i=0; i<3997; i++)
  w[i]=w[i]+z[i];
}
```

Max. parallelism (no fusion) →

```
doall (i=0; i<400; i++)
 doall (j=0; j<3997; j++)
  z_e[j][i]=0
doall (i=0; i<400; i++)
 doall (j=0; j<3997; j++)
  for (k=0; k<4000; k++)
   z_e[j][i]=z_e[j][i]+B[j][k]*x[i][k];
doall (i=0; i<3997; i++)
 for (j=0; j<400; j++)
  w[i]=w[i]+z_e[i][j];
doall (i=0; i<3997; i++)
 z[i] = z_e[i][399];
```

Data accumulation

→ 2 levels of parallelism, but poor data reuse (on array z_e)

Parallelism/Locality Tradeoff Example (cont.)

```
/*
 * Original code:
 *   Simplified CSLC-LMS
 */
for (k=0; k<400; k++) {
  for (i=0; i<3997; i++) {
    z[i]=0;
    for (j=0; j<4000; j++)
      z[i]= z[i]+B[i][j]*x[k][j];
  }
  for (i=0; i<3997; i++)
    w[i]=w[i]+z[i];
}
```

Max. fusion →

```
doall (i=0; i<3997; i++)
  for (j=0; j<400; j++) {
    z[i]=0;
    for (k=0; k<4000; k++)
      z[i]=z[i]+B[i][k]*x[j][k];
    w[i]=w[i]+z[i];
  }
```

Aggressive loop fusion destroys parallelism (i.e., only 1 degree of parallelism)

→ Very good data reuse (on array z), but only 1 level of parallelism

Parallelism/Locality Tradeoff Example (cont.)

```
/*
 * Original code:
 *   Simplified CSLC-LMS
 */
for (k=0; k<400; k++) {
  for (i=0; i<3997; i++) {
    z[i]=0;
    for (j=0; j<4000; j++)
      z[i]= z[i]+B[i][j]*x[k][j];
  }
  for (i=0; i<3997; i++)
    w[i]=w[i]+z[i];
}
```

Parallelism with partial fusion →

Expansion of array z

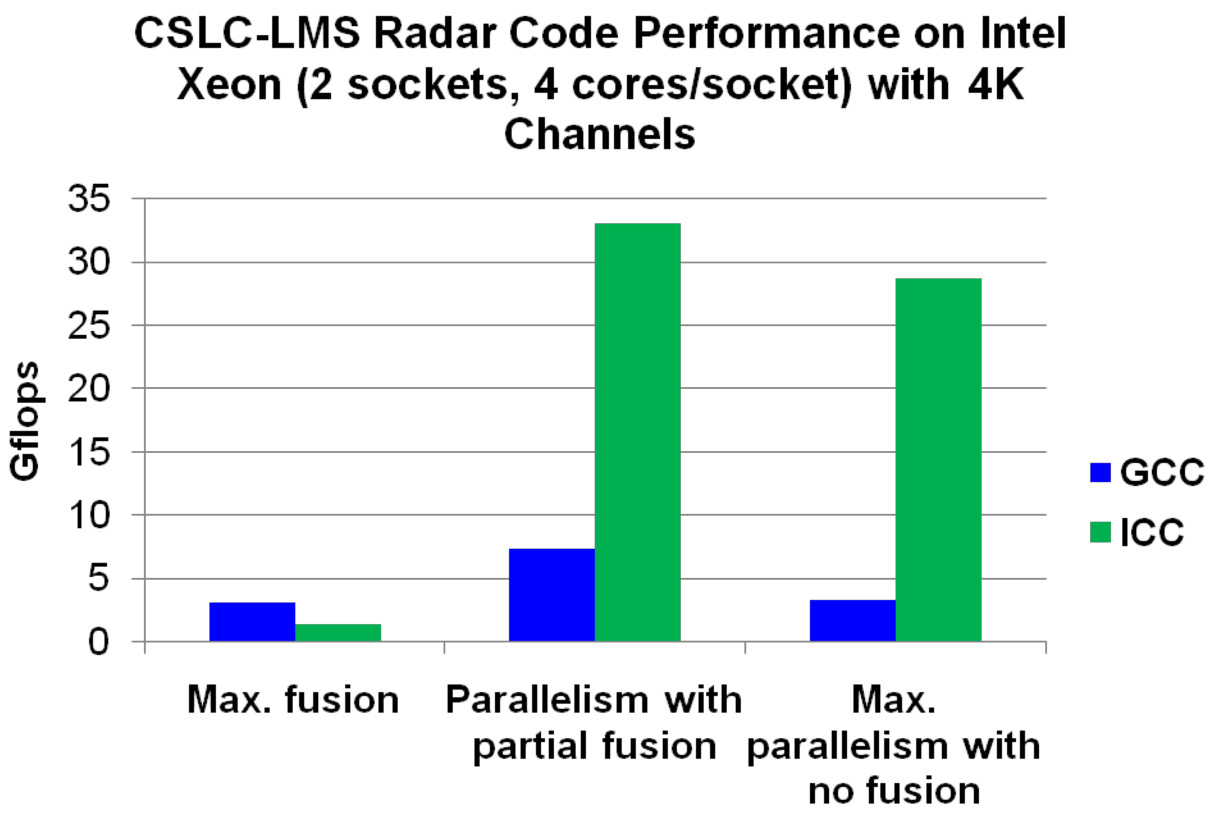
Partial fusion doesn't decrease parallelism

```
doall (i=0; i<3997; i++) {
  doall (j=0; j<400; j++) {
    z_e[i][j]=0;
    for (k=0; k<4000; k++)
      z_e[i][j]=z_e[i][j]+B[i][k]*x[j][k];
  }
  for (j=0; j<400; j++)
    w[i]=w[i]+z_e[i][j];
}
doall (i=0; i<3997; i++)
  z[i]=z_e[i][399];
```

Data accumulation

→ 2 levels of parallelism with good data reuse (on array z_e)

Parallelism/Locality Tradeoffs: Performance Numbers



→ **Code with a good balance between parallelism and fusion performs best**

R-Stream: Affine Scheduling and Fusion

R-Stream uses a heuristic based on an *objective function* with several *cost coefficients*:

- slowdown in execution if a loop *p* is executed sequentially rather than in parallel
- cost in performance if two loops *p* and *q* remain unfused rather than fused

$$minimize \left(\sum_{l \in \text{loops}} w_l \, p_l + \sum_{e \in \text{loop edges}} u_e \, f_{\mathrm{e}} \right)$$

slowdown in sequential execution

cost of unfusing two loops

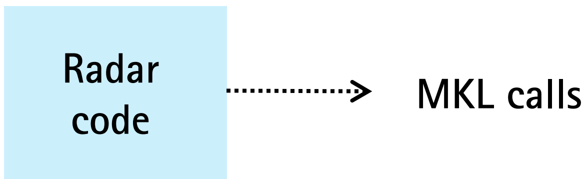
These two cost coefficients address parallelism and locality in a *unified and unbiased manner* (as opposed to traditional compilers)

Fine-grained parallelism, such as SIMD, can also be modeled using similar formulation

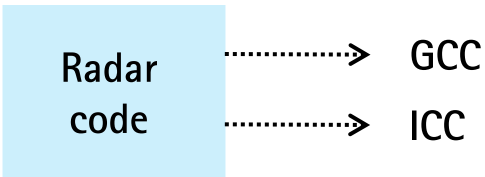
Patent Pending

Experimental Evaluation

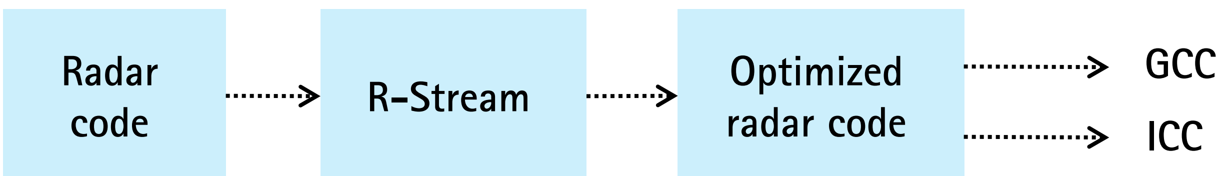
Configuration 1: MKL



Configuration 2: Low-level compilers



Configuration 3: R-Stream



Main comparisons:

- R-Stream High-Level C Compiler 3.1.2
- Intel MKL 10.2.1

Experimental Evaluation (cont.)

Intel Xeon workstation:

- Dual quad-core E5405 Xeon processors (8 cores total)
- 9GB memory

8 OpenMP threads

Single precision floating point data

Low-level compilers and the used flags:

- GCC: -O6 -fno-trapping-math -ftree-vectorize -msse3 -fopenmp
- ICC: -fast -openmp

Radar Benchmarks

Beamforming algorithms:

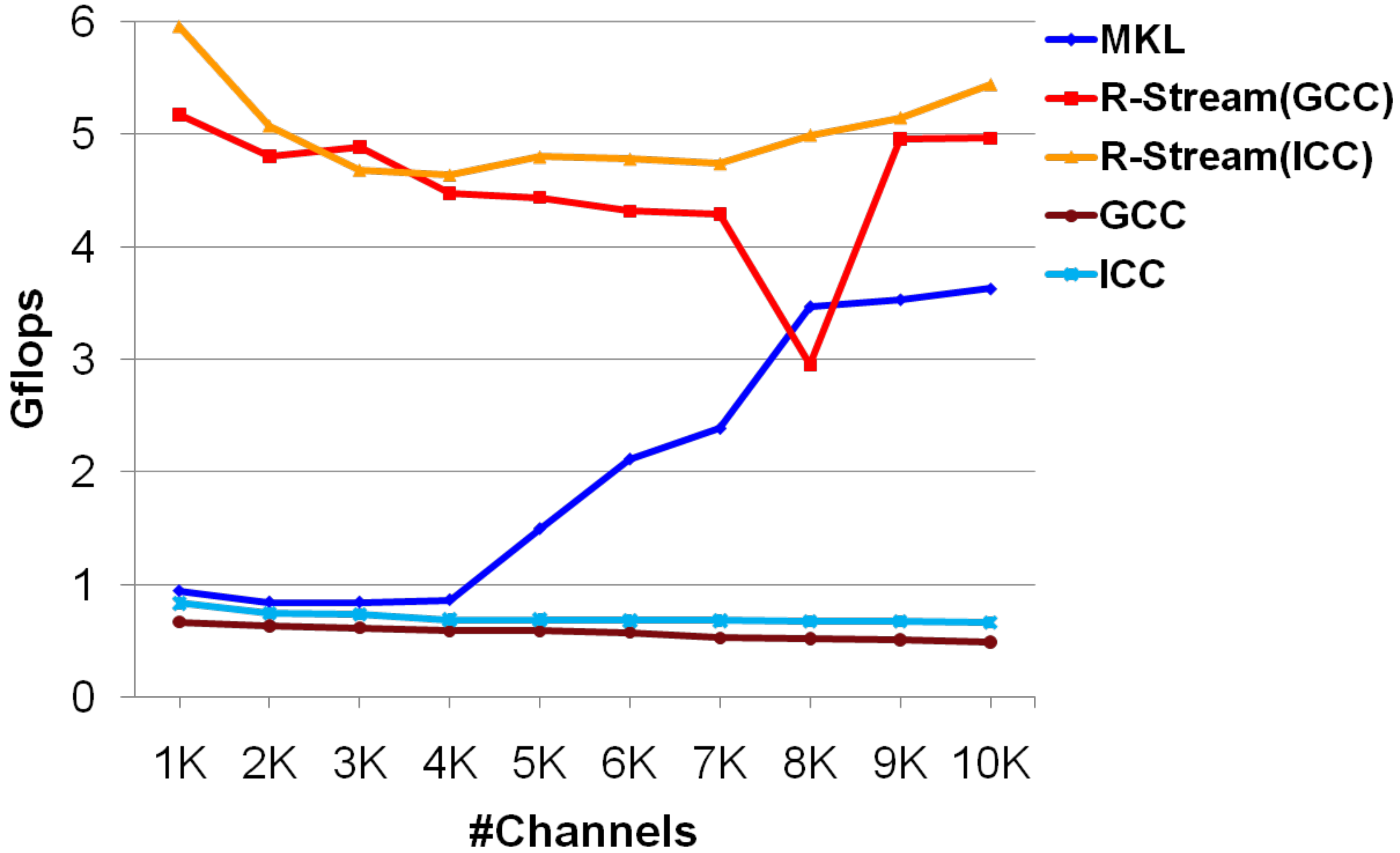
- **MVDR-SER:** Minimum Variance Distortionless Response using Sequential Regression
- **CSLC-LMS:** Coherent Sidelobe Cancellation using Least Mean Square
- **CSLC-RLS:** Coherent Sidelobe Cancellation using Robust Least Square

Expressed in sequential ANSI C

400 radar iterations

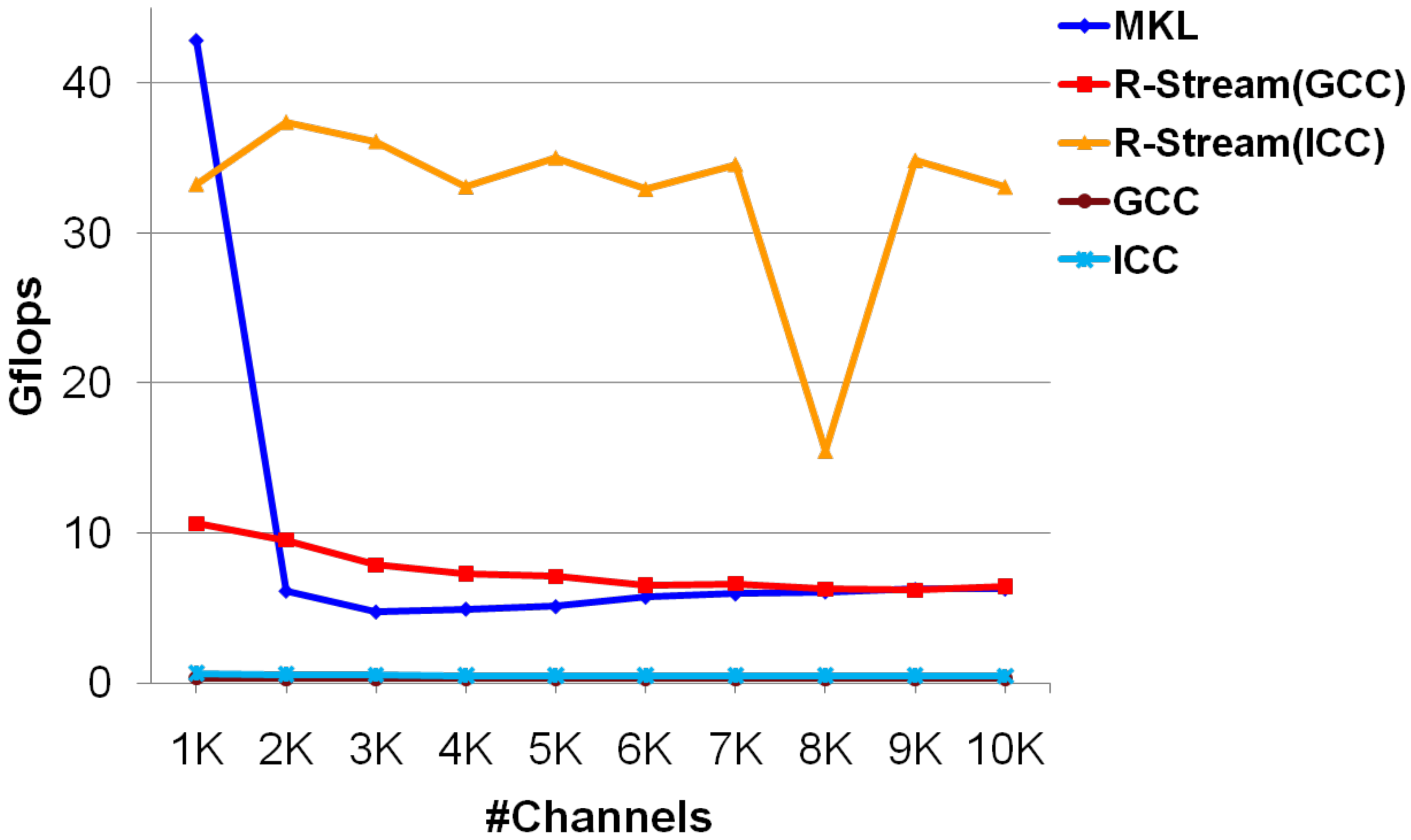
Compute 3 radar sidelobes (for CSLC-LMS and CSLC-RLS)

MVDR-SER



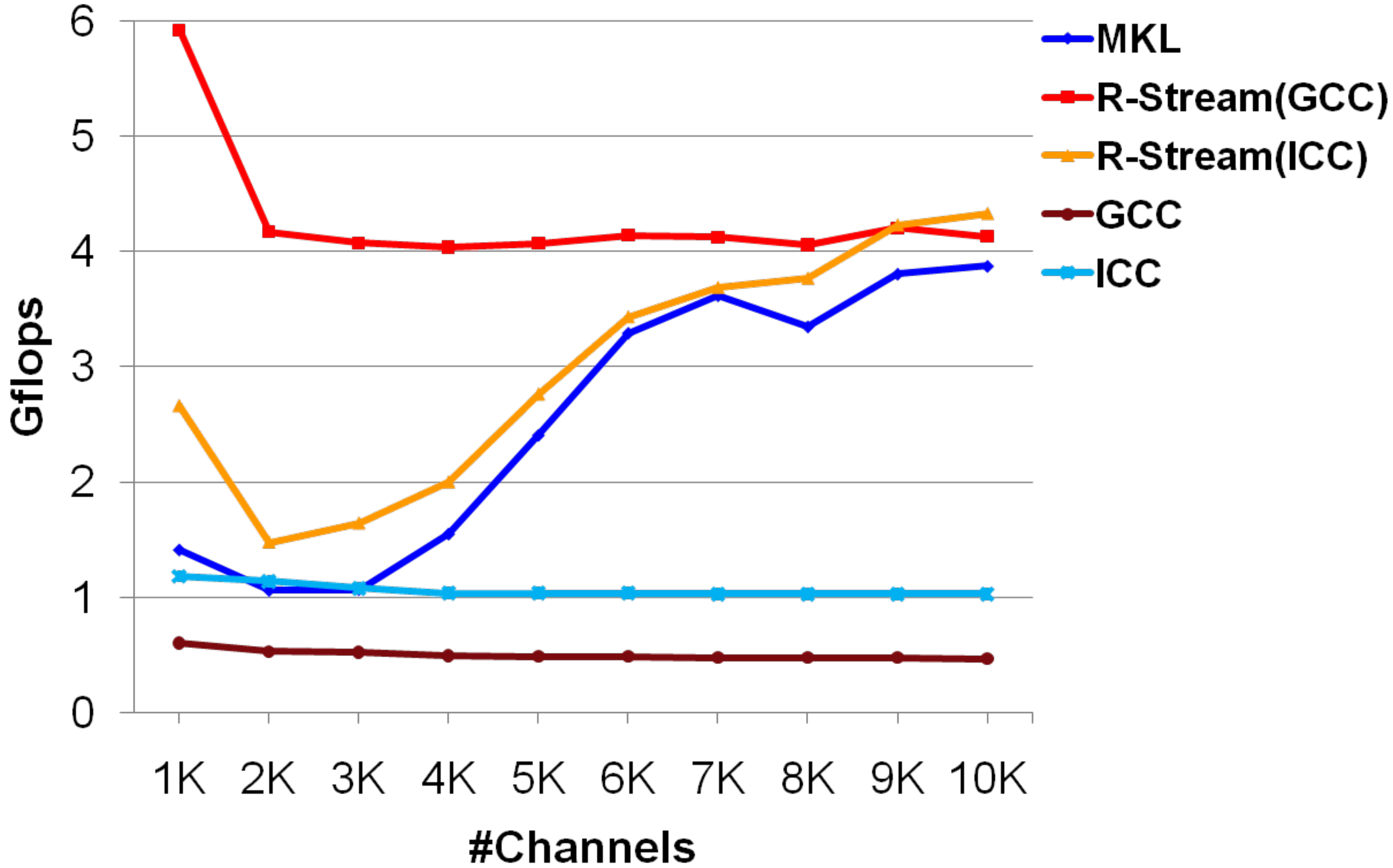
6
5
4
3
2
1
0
Gflops
1K
2K
3K
4K
5K
6K
7K
8K
9K
10K
#Channels
MKL
R-Stream(GCC)
R-Stream(ICC)
GCC
ICC

CSLC-LMS



MKL
R-Stream(GCC)
R-Stream(ICC)
GCC
ICC
Gflops
0
10
20
30
40
1K
2K
3K
4K
5K
6K
7K
8K
9K
10K
#Channels

CSLC-RLS



6
5
4
3
2
1
0
Gflops
1K
2K
3K
4K
5K
6K
7K
8K
9K
10K
#Channels
MKL
R-Stream(GCC)
R-Stream(ICC)
GCC
ICC

Conclusions

R-Stream performs automatic parallelization for beamforming algorithms

- Sequential ANSI C inputs

Optimizations not just for parallelism but also locality

- A unified approach to precise tradeoffs between parallelism and locality

Performance results very good

- Can be *up to 7x faster* than implementations based on the leading industrial math library (Intel MKL)