Abstract

VSIPL++: Parallel Performance

HPEC 2004

CodeSourcery, LLC

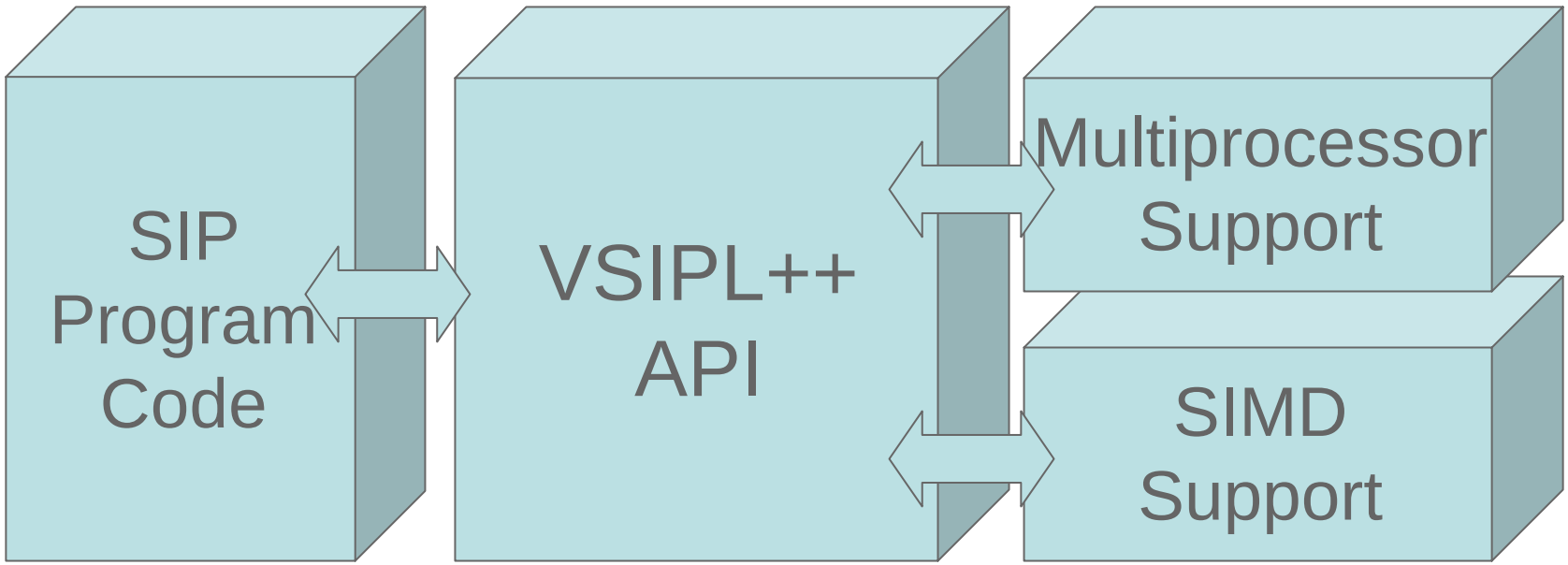
September 30, 2004

CodeSourcery LLC

Challenge

- "Object oriented technology reduces software cost."
- "Fully utilizing HPEC systems for SIP applications requires managing operations at the lowest possible level."
- "There is great concern that these two approaches may be fundamentally at odds."

Parallel Performance Vision



"Drastically reduce the performance penalties associated with deploying object-oriented software on high performance parallel embedded systems."

"Automated to reduce implementation cost."

Advantages of VSIPL

- Portability
  - Code can be reused on any system for which a VSIPL implementation is available.
- Performance
  - Vendor-optimized implementations perform better than most handwritten code.
- Productivity
  - Reduces SLOC count.
  - Code is easier to read.
  - Skills learned on one project are applicable to others.
  - Eliminates use of assembly code.

Limitations of VSIPL

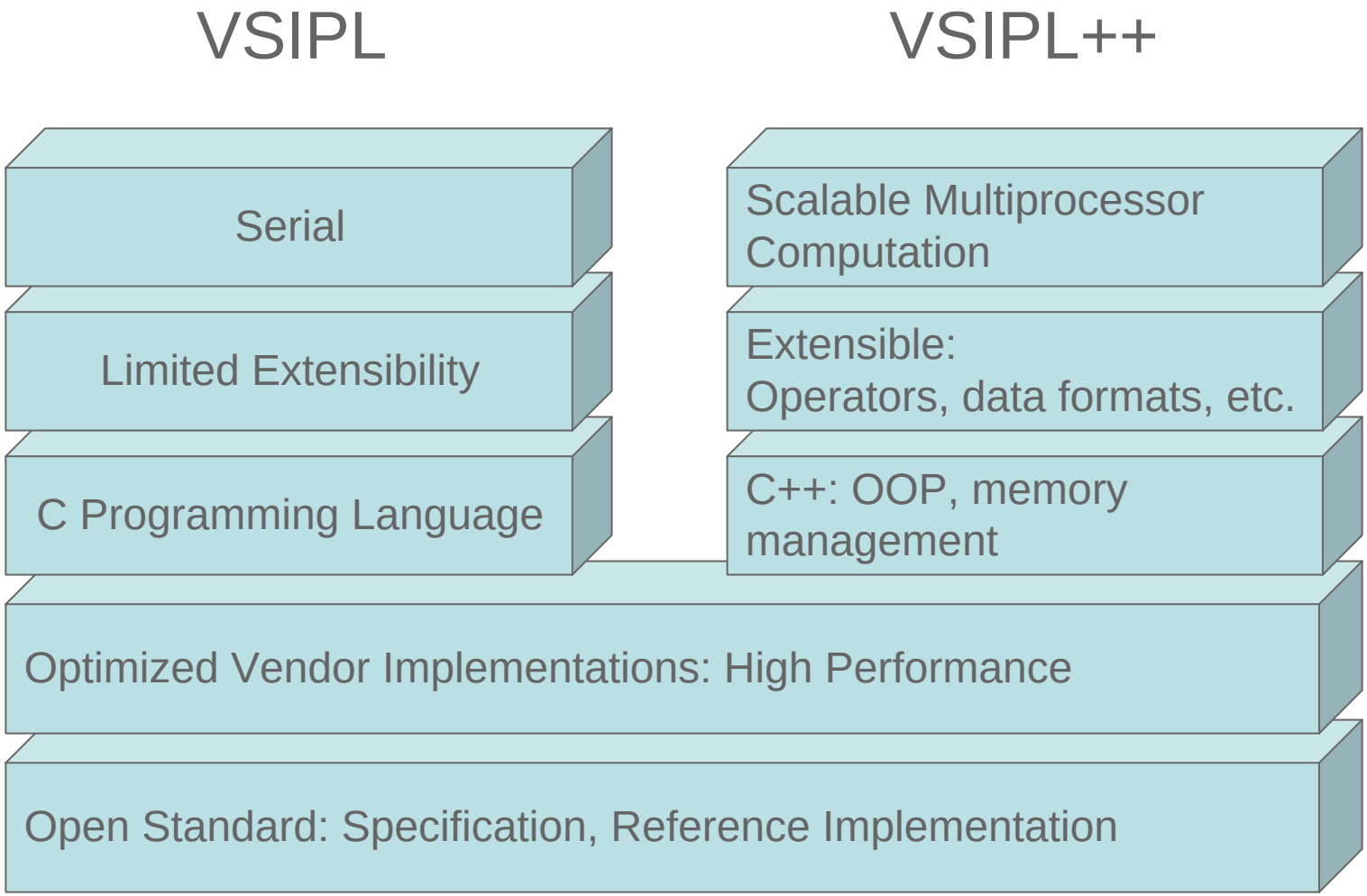
- Uses C Programming Language
  - "Modern object oriented languages (e.g., C++) have consistently reduced the development time of software projects."
  - Manual memory management.
  - Cumbersome syntax.
- Inflexible
  - Abstractions prevent users from adding new high-performance functionality.
  - No provisions for loop fusion.
  - No way to avoid unnecessary block copies.
- Not Scalable
  - No support for MPI or threads.
  - SIMD support must be entirely coded by vendor; user cannot take advantage of SIMD directly.

Parallelism: Current Practice

MPI used for communication, but:

- MPI code often a significant fraction of total program code.
- MPI code notoriously hard to debug.
- Tendency to hard-code number of processors, data sizes, etc.
  - Reduces portability!
- Conclusion: users should specify only data layout.

Atop VSIPL's Foundation



Leverage VSIPL Model

- Same terminology:
  - Blocks store data.
  - Views provide access to data.
  - Etc.
- Same basic functionality:
  - Element-wise operations.
  - Signal processing.
  - Linear algebra.

VSIPL++ Status

- Serial Specification: Version 1.0a
  - Support for all functionality of VSIPL.
  - Flexible block abstraction permits varying data storage formats.
  - Specification permits loop fusion, efficient use of storage.
  - Automated memory management.
- Reference Implementation: Version 0.95
  - Support for functionality in the specification.
  - Used in several demo programs — see next talks.
  - Built atop VSIPL reference implementation for maximum portability.
- Parallel Specification: Version 0.5
  - High-level design complete.

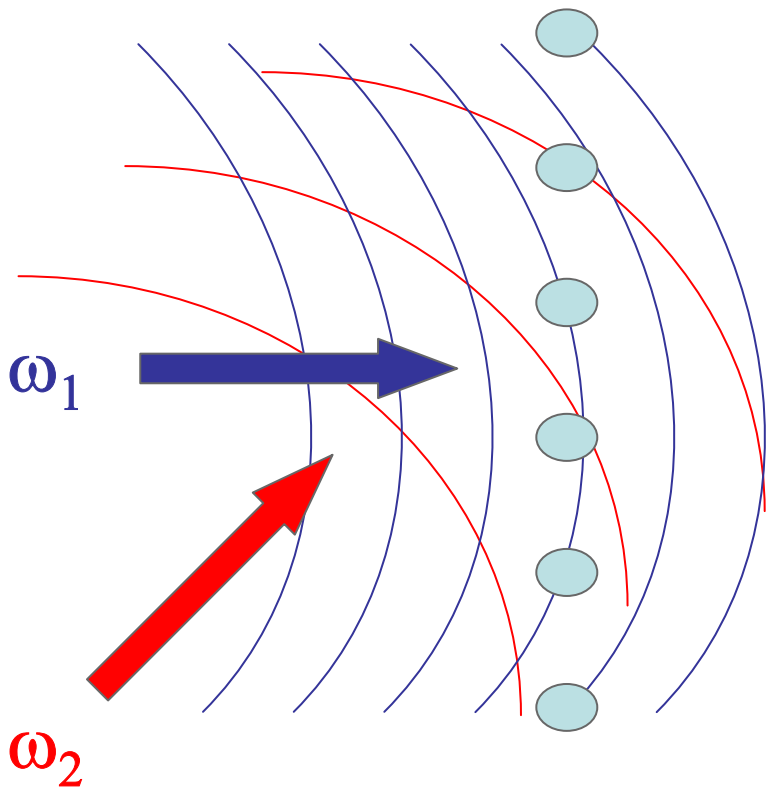
k-Ω Beamformer

Input:

- Noisy signal arriving at a row of uniformly distributed sensors.

Output:

- Bearing and frequency of signal sources.



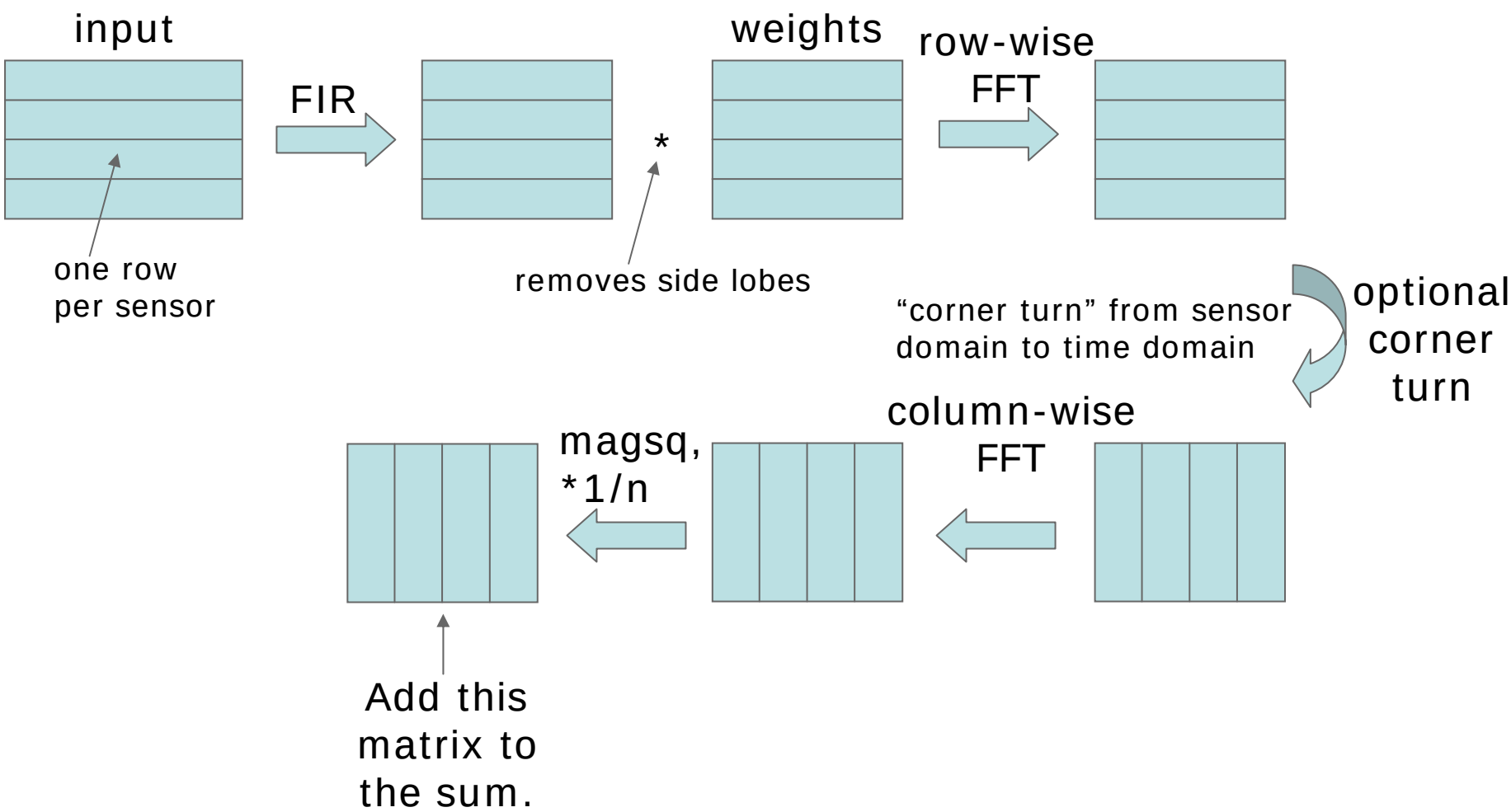
SIP Primitives Used

- Computation:
  - FIR filters
  - Element-wise operations (e.g, magsq)
  - FFTs
  - Minimum/average values
- Communication:
  - Corner-turn
    - All-to-all communication
  - Minimum/average values
    - Gather

Computation

1. Filter signal to remove high-frequency noise. (FIR)
2. Remove side-lobes resulting from discretization of data. (mult)
3. Apply Fourier transform in time domain. (FFT)
4. Apply Fourier transform in space domain. (FFT)
5. Compute power spectra. (mult, magsq)

Diagram of the Kernel



VSIPL Kernel

Seven statements required:

```
for (i = n; i > 0; --i) {
  filtered = filter (firs, signal);
  vsip_mmul_f (weights, filtered, filtered);
  vsip_rcfftmpop_f (space_fft, filtered,
                    fft_output);
  vsip_ccfftmpi_f (time_fft, fft_output);
  vsip_mcmagsq_f (fft_output, power);
  vsip_ssmul_f (1.0 / n, power);
  vsip_madd_f (power, spectra, spectra);
}
```

VSIPL++ Kernel

One statement required:

```
for (i = n; i > 0; --i)
  spectra += 1/n *
    magsq (
     time_fft (space_fft (weights *
                          filter (firs,
                                  signal)));
```

No changes are required for distributed operation.

Distribution in User Code

Serial case:

```
Matrix<float_t, Dense<2, float_t> >
   signal_matrix;
```

Parallel case:

```
typedef Dense<2, float_t> subblock;
typedef Distributed<2, float_t, subblock, ROW>
  Block2R_t;
Matrix<float_t, Block2R_t> signal_matrix;
```

User writes no MPI code.

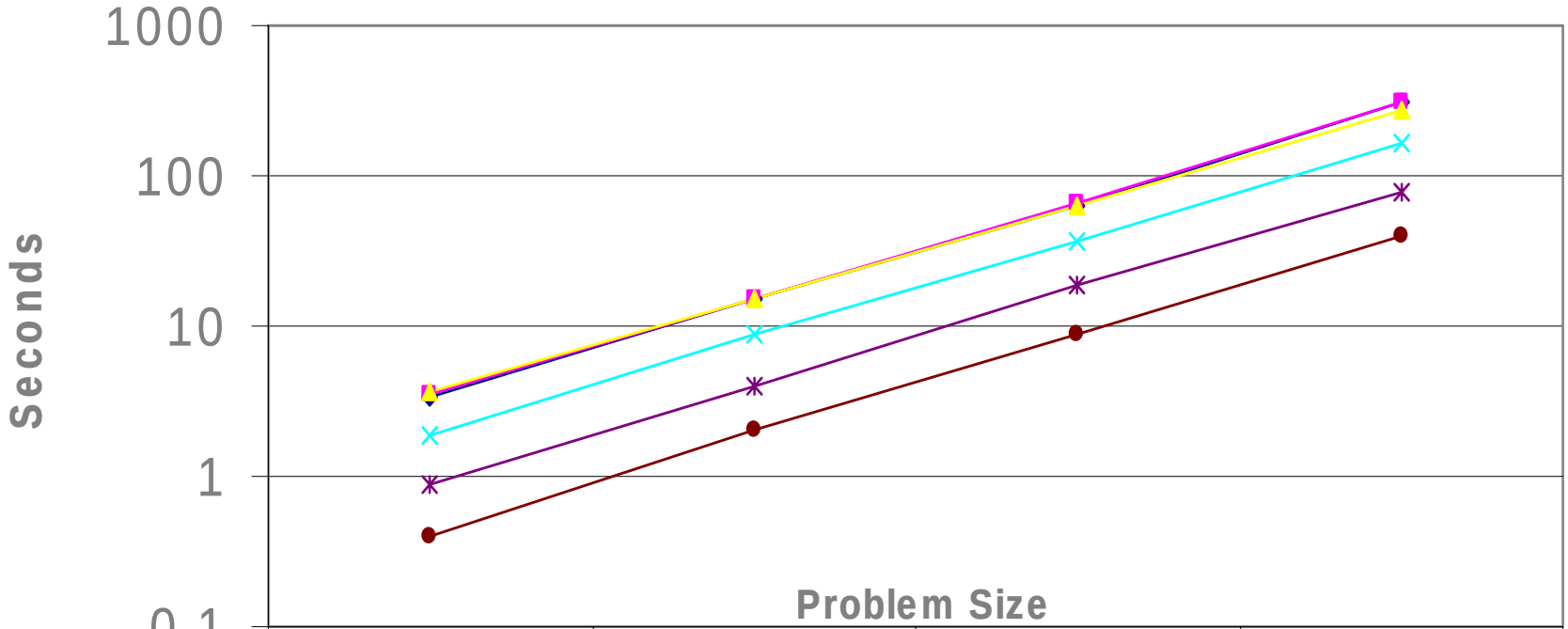
VSIPL++ Implementation

- Added `DistributedBlock`:
  - Uses a "standard" VSIPL++ block on each processor.
  - Uses MPI routines for communication when performing block assignment.
- Added specializations:
  - FFT, FIR, etc. modified to handle `DistributedBlock`.

Performance Measurement

- Test system:
  - AFRL HPC system
  - 2.2GHz Pentium 4 cluster
- Measured only main loop
  - No input/output
- Used Pentium Timestamp Counter
- MPI All-to-all not included in timings
  - Accounts for 10-25%

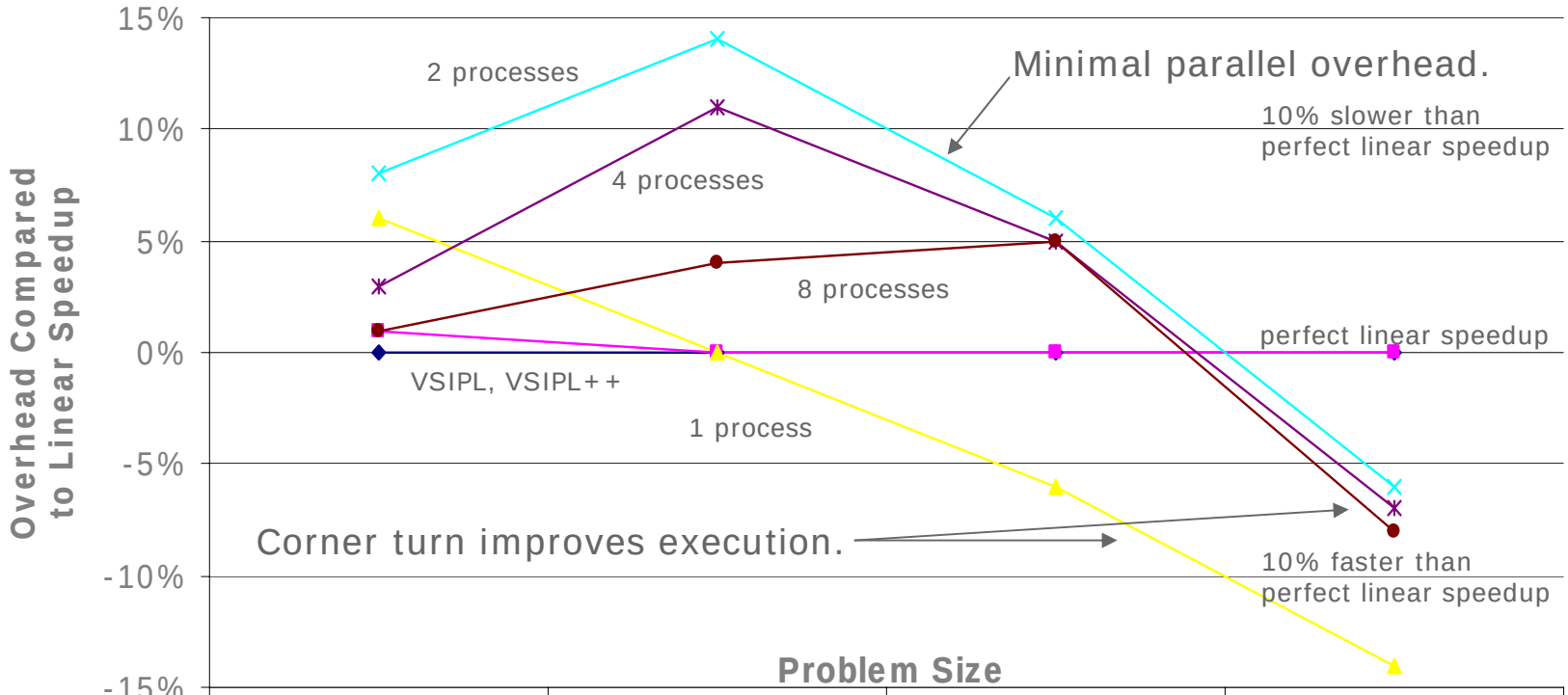
VSIPL++ Performance



	256x512	512x1024	1024x2048	2048x4096
VSIPL	3.3	15	64	306
VSIPL++	3.5	15	66	314
|| VSIPL++ (1)	3.6	15	63	277
|| VSIPL++ (2)	1.9	9	37	165
|| VSIPL++ (4)	0.9	4	19	78
|| VSIPL++ (8)	0.4	2	9	40

CodeSourcery LLC

Parallel Speedup



	256x512	512x1024	1024x2048	2048x4096
VSIPL	0%	0%	0%	0%
VSIPL++	1%	0%	0%	0%
|| VSIPL++ (1)	6%	0%	-6%	-14%
|| VSIPL++ (2)	8%	14%	6%	-6%
|| VSIPL++ (4)	3%	11%	5%	-7%
|| VSIPL++ (8)	1%	4%	5%	-8%

CodeSourcery LLC

Conclusions

- VSIPL++ imposes no overhead:
  - VSIPL++ performance nearly identical to VSIPL performance.
- VSIPL++ achieves near-linear parallel speedup:
  - No tuning of MPI, VSIPL++, or application code.
- Absolute performance limited by VSIPL implementation, MPI implementation, compiler.

VSIPL++

Visit the HPEC-SI website

http://www.hpec-si.org

- for VSIPL++ specifications
- for VSIPL++ reference implementation
- to participate in VSIPL++ development

VSIPL++: Parallel Performance

HPEC 2004

CodeSourcery, LLC

September 30, 2004