PVTOL: A High-Level Signal Processing Library for Multicore Processors

Hahn Kim, Nadya Bliss, Ryan Haney, Jeremy Kepner, Matthew Marzilli, Sanjeev Mohindra, Sharon Sacco, Glenn Schrader, Edward Rutledge

HPEC 2007

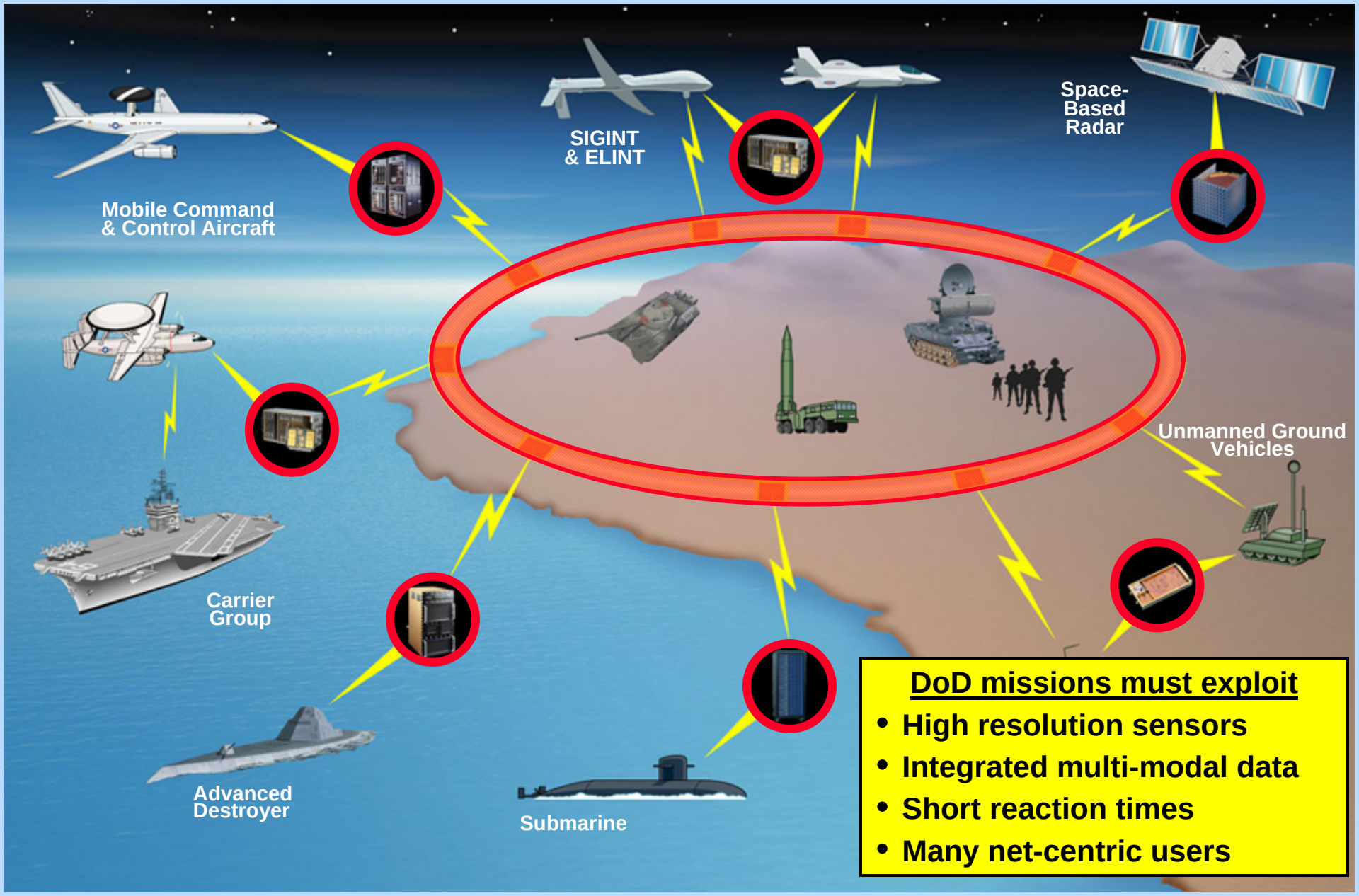
20 September 2007

This work is sponsored by the Department of the Air Force under Air Force contract FA8721-05-C-0002. Opinions, interpretations, conclusions and recommendataions are those of the author and are not necessarily endorsed by the United States Government.

Outline

- **Background**
  - Motivation
  - Multicore Processors
  - Programming Challenges
- **Parallel Vector Tile Optimizing Library**
- **Results**
- **Summary**

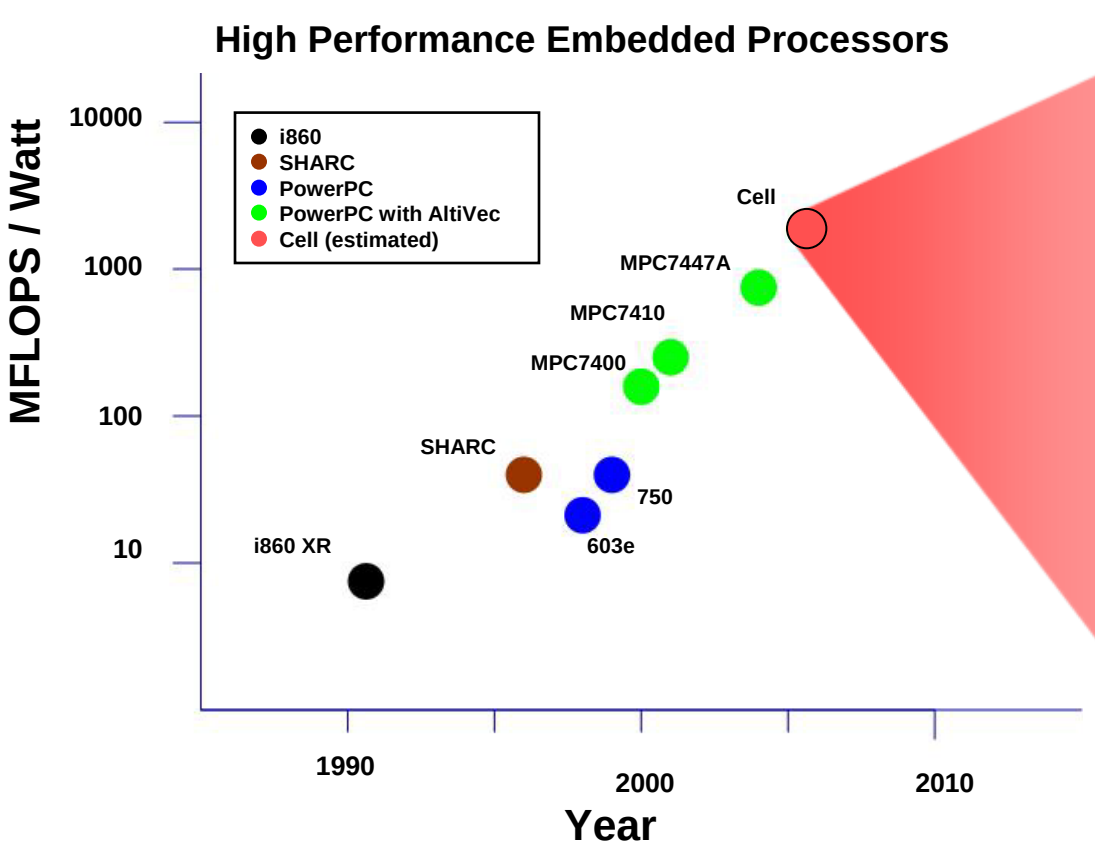
Future DoD Sensor Missions

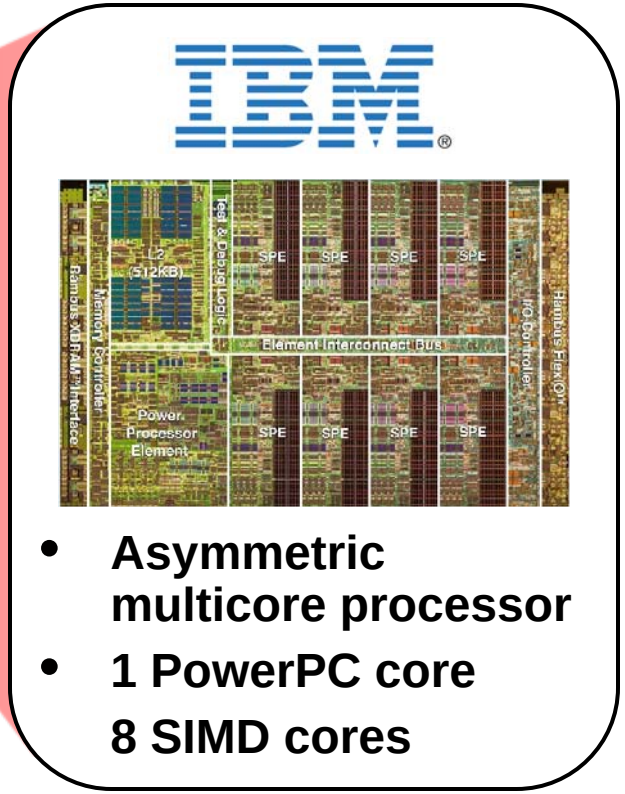


DoD missions must exploit

- High resolution sensors
- Integrated multi-modal data
- Short reaction times
- Many net-centric users

Embedded Processor Evolution



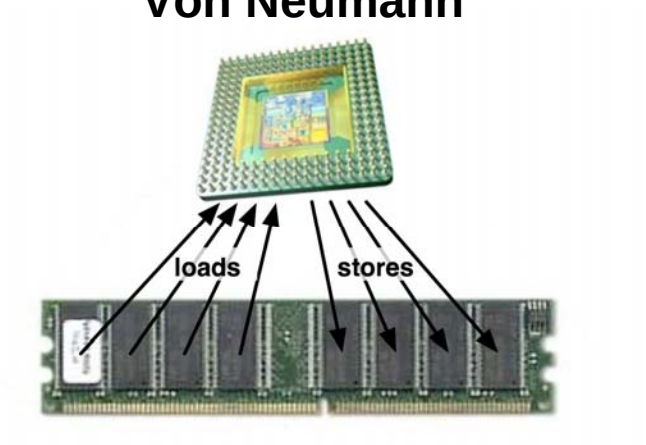


- Asymmetric multicore processor
- 1 PowerPC core
  8 SIMD cores

- 20 years of exponential growth in FLOPS / Watt
- Requires switching architectures every ~5 years
- Cell processor is current high performance architecture

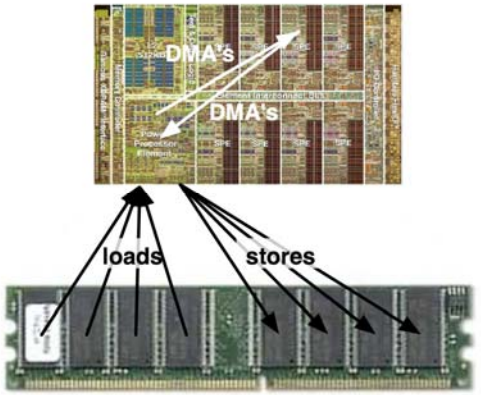
Multicore Programming Challenge

Past Programming Model: Von Neumann



- Great success of Moore's Law era
  - Simple model: load, op, store
  - Many transistors devoted to delivering this model
- Moore's Law is ending
  - Need transistors for performance

Future Programming Model: ???



- Processor topology includes:
  - Registers, cache, local memory, remote memory, disk
- Multicore processors have *multiple* programming models

Increased performance at the cost of exposing complexity to the programmer

Example: Time-Domain FIR

ANSI C

```
for (i = K; i > 0; i--) {

  /* Set accumulators and pointers for dot
   * product for output point */
  r1 = Rin;
  r2 = Iin;
  o1 = Rout;
  o2 = Iout;

  /* Calculate contributions from a single
   * kernel point */
  for (j = 0; j < N; j++) {

    *o1 += *k1 * *r1 - *k2 * *r2;
    *o2 += *k2 * *r1 + *k1 * *r2;

    r1++; r2++; o1++; o2++;
  }

  /* Update input pointers */
  k1++; k2++;
  Rout++;
  Iout++;
}
```

- **ANSI C is easy to understand**
- **Cell increases complexity:**
  - **Communication requires synchronization**
  - **Computation requires SIMD**

Cell Manager C (Communication)

```
/* Fill input vector and filter buffers and send to workers*/
mcf_m_tile_channel_get_buffer(..., &vector_in, ...);
mcf_m_tile_channel_get_buffer(..., &filter, ...);
init_buffer(&vector_in);
init_buffer(&filter);
mcf_m_tile_channel_put_buffer(..., &vector_in, ...);
mcf_m_tile_channel_put_buffer(..., &filter, ...);

/* Wait for worker to process and get a full output vector buffer */
mcf_m_tile_channel_get_buffer(..., &vector_out, ...);
```

Cell Worker C (Communication)

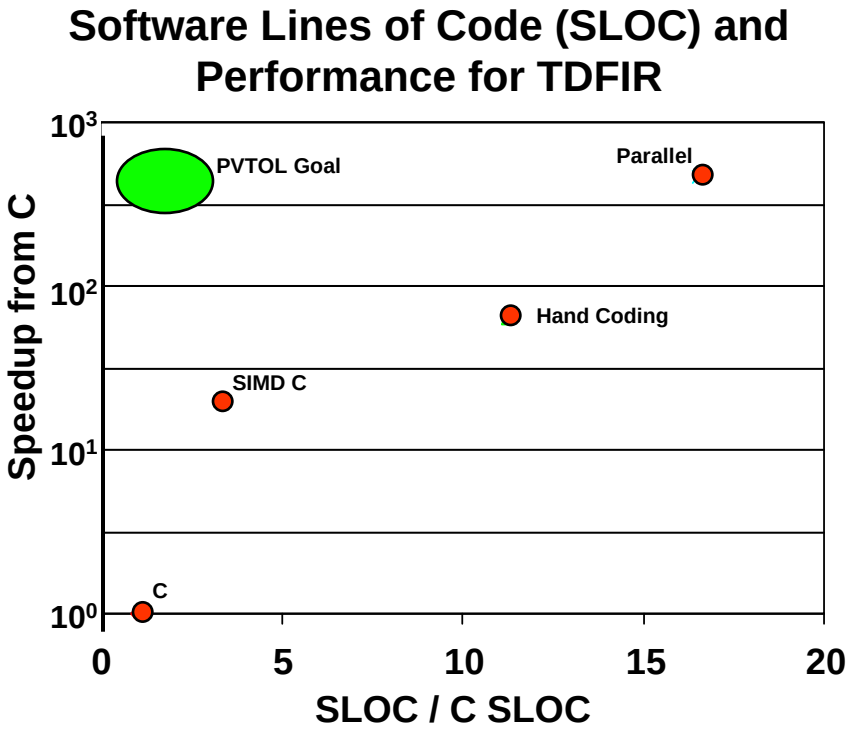
```
while (mcf_w_tile_channel_is_not_end_of_channel(...)) {
   /* Get buffers */
   mcf_w_tile_channel_get_buffer(..., &vector_in, ...);
   mcf_w_tile_channel_get_buffer(..., &filter, ...);
   mcf_w_tile_channel_get_buffer(..., &vector_out ...);

   /* Perform the filter operation */
   filter_vect(vector_in, filter, vector_out);

   /* Send results back to manager */
   mcf_w_tile_channel_put_buffer(..., &vector_out, ...);

   /* Put back input vector and filter buffers */
   mcf_w_tile_channel_put_buffer(..., &filter, ...);
   mcf_w_tile_channel_put_buffer(..., &vector_in, ...);
}
```

Cell Worker C (Computation)

```
/* Load reference data and shift  */
ir0 = *Rin++;
ii0 = *Iin++;
ir1 = (vector float) spu_shuffle(irOld, ir0, shift1);
ii1 = (vector float) spu_shuffle(iiOld, ii0, shift1);

Rtemp = spu_madd(kr0, ir0, Rtemp); Itemp = spu_madd(kr0, ii0, Itemp);
Rtemp = spu_nmsub(ki0, ii0, Rtemp); Itemp = spu_madd(ki0, ir0, Itemp);
```

Example: Time-Domain FIR

Performance vs. Effort



	C	SIMD C	Hand Coding	Parallel (8 SPE)
Lines of Code	33	110	371	546
Performance Efficiency (1 SPE)	0.014	0.27	0.88	0.82
GFLOPS (2.4 GHz)	0.27	5.2	17	126

PVTOL Goal: Achieve high performance with little effort

S. Sacco, et al. "Exploring the Cell with HPEC Challenge Benchmarks." HPEC 2006.

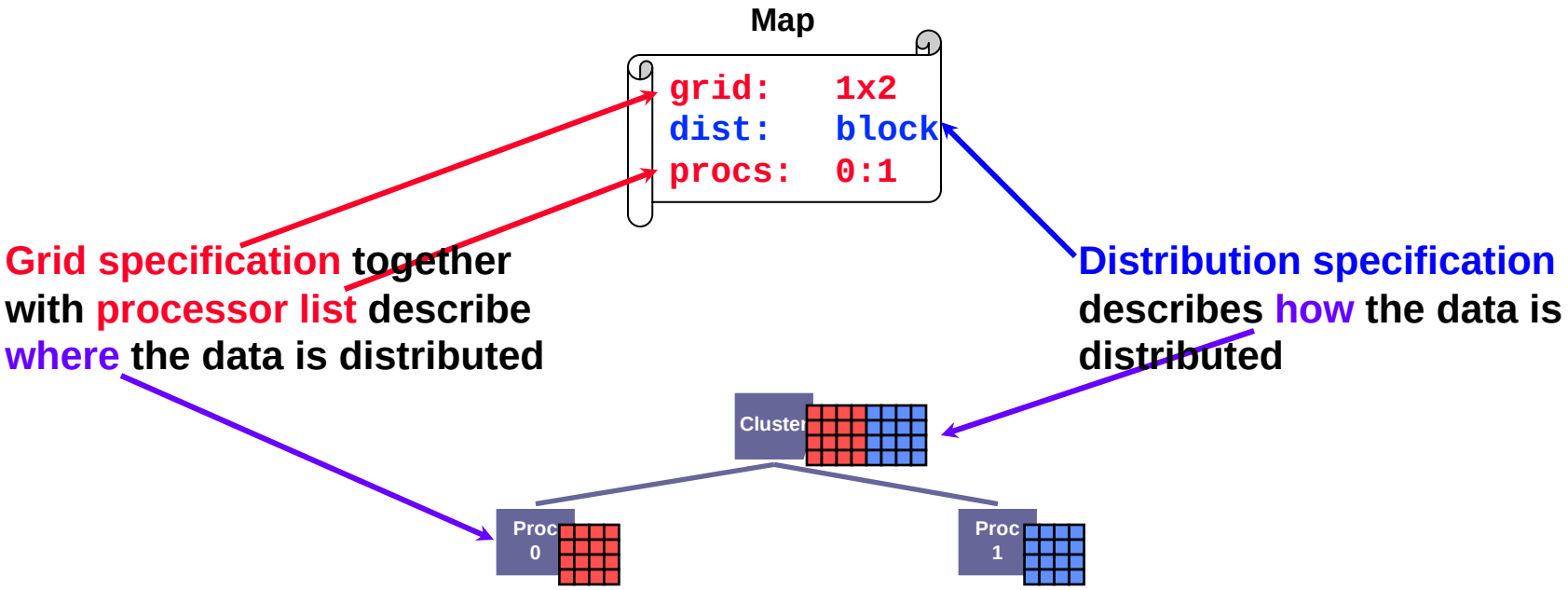
Outline

- Background
- Parallel Vector Tile Optimizing Library
  - Map-Based Programming
  - PVTOL Architecture
- Results
- Summary

Map-Based Programming

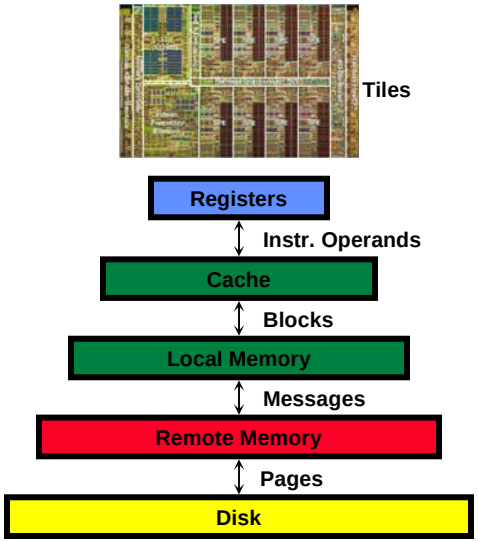
- A map is an assignment of blocks of data to processing elements
- Maps have been demonstrated in several technologies

Technology	Organization	Language	Year
Parallel Vector Library	MIT-LL	C++	2000
pMatlab	MIT-LL	MATLAB	2003
VSIPL++	HPEC-SI	C++	2006

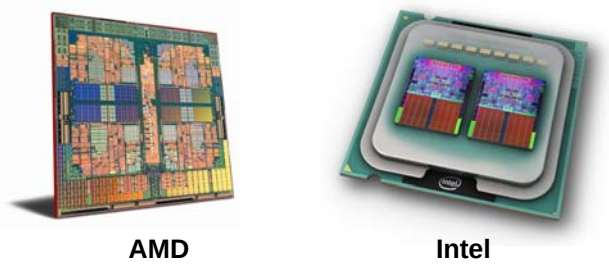


New Challenges

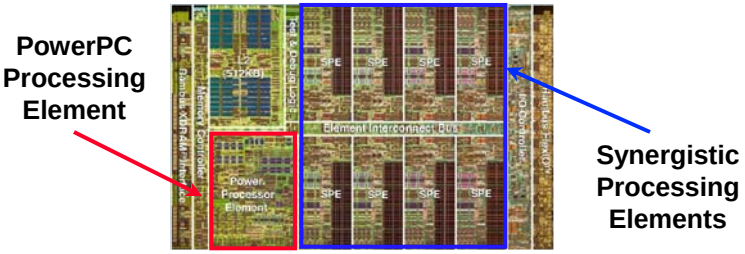
- **Hierarchy**
  - Extend maps to support the entire storage hierarchy



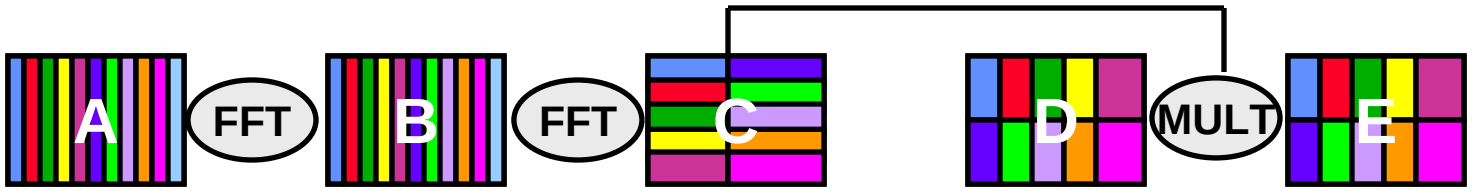
- **Heterogeneity**
  - Different architectures *between* processors



  - Different architectures *within* a processor

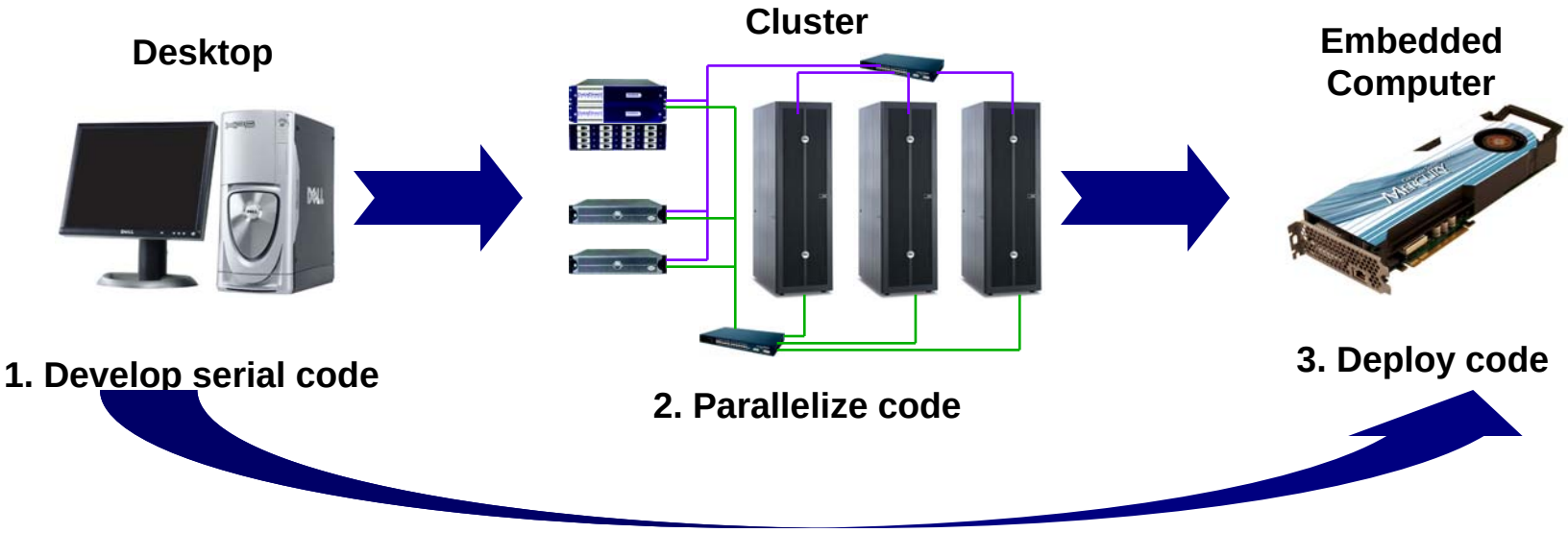


- **Automated Mapping**
  - Allow maps to be constructed using automated techniques



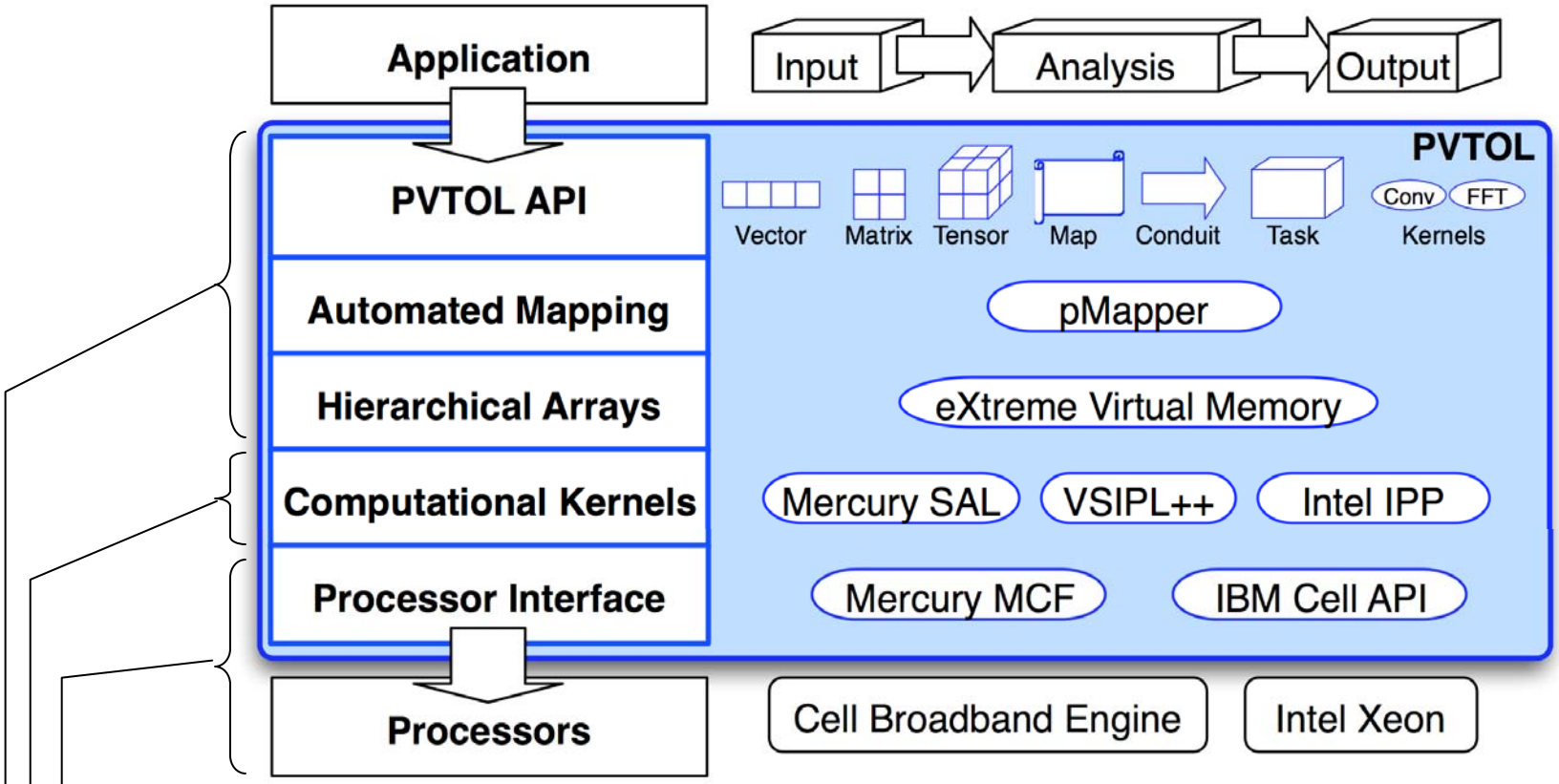
PVTOL Goals

- PVTOL is a portable and scalable middleware library for multicore processors
- Enables incremental development



Make parallel programming as easy as serial programming

PVTOL Architecture



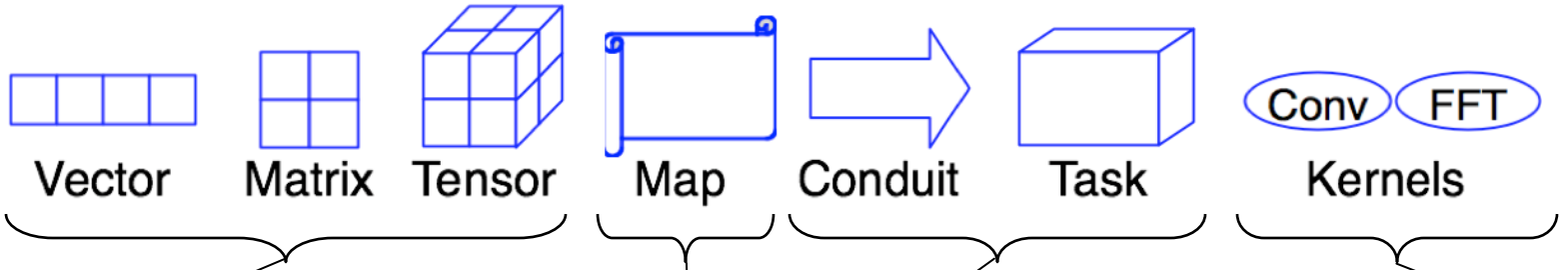
Portability: Runs on a range of architectures

Performance: Achieves high performance

Productivity: Minimizes effort at user level

PVTOL preserves the simple load-store programming model in software

PVTOL API



Data structures encapsulate data allocated across the storage hierarchy into objects

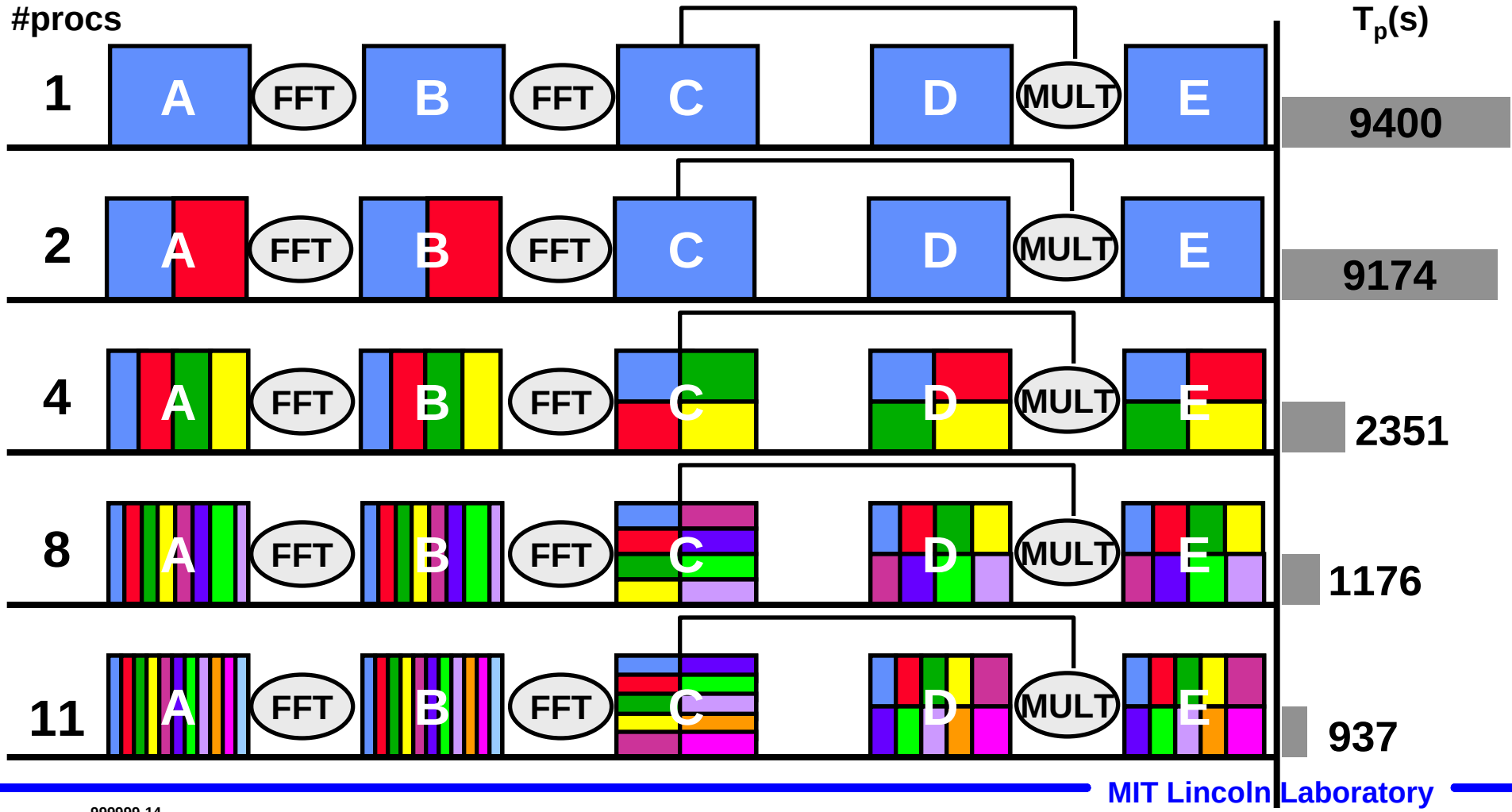
Tasks encapsulate computation. Conduits pass data between tasks

Maps describe how to assign blocks of data across the storage hierarchy

Kernel objects/functions encapsulate common operations. Kernels operate on PVTOL data structures

Automated Mapping

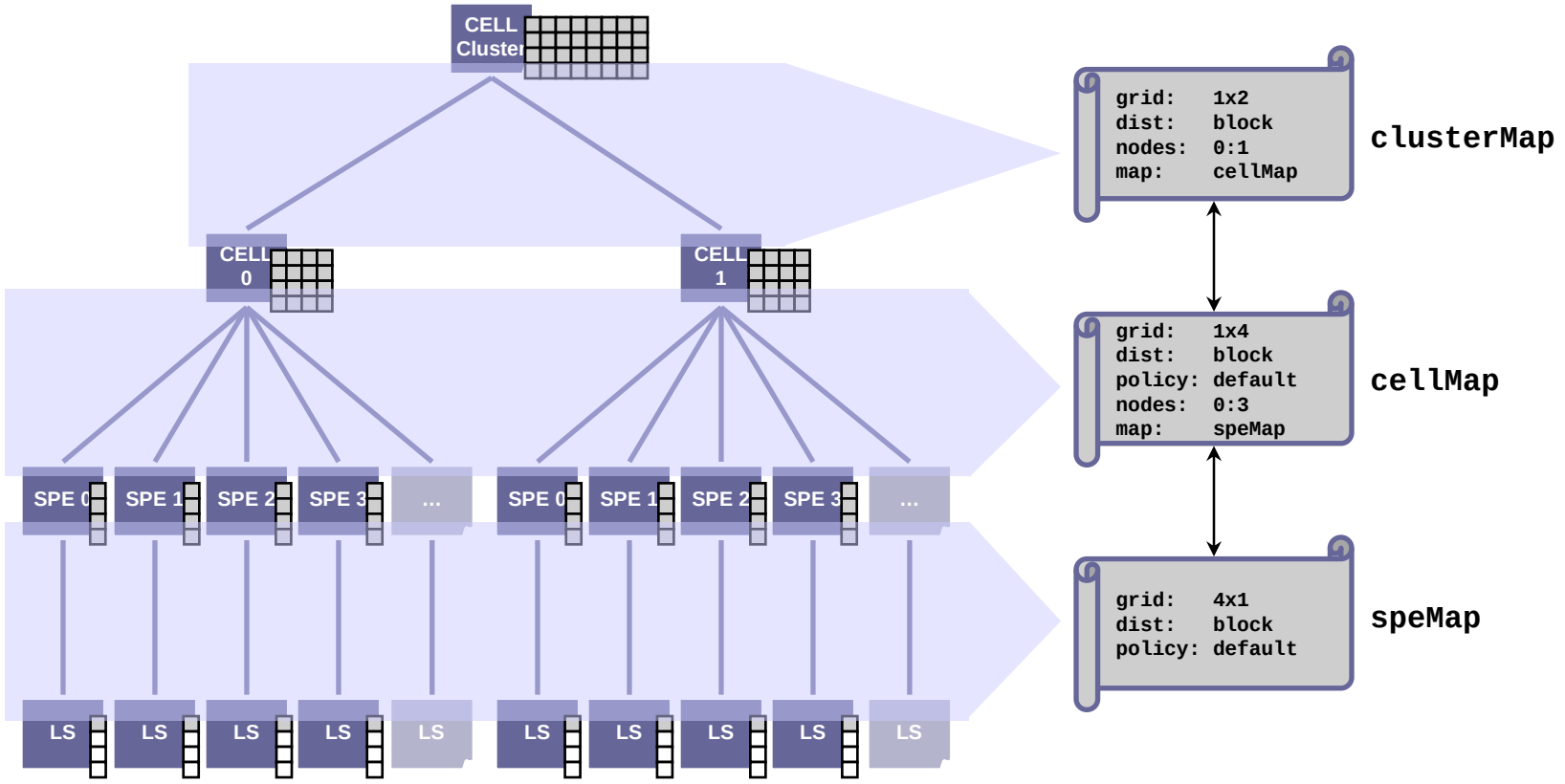
- Simulate the architecture using pMapper simulator infrastructure
- Use pMapper to automate mapping and predict performance



N. Bliss, et al. "Automatic Mapping of the HPEC Challenge Benchmarks." HPEC 2006.

Hierarchical Arrays

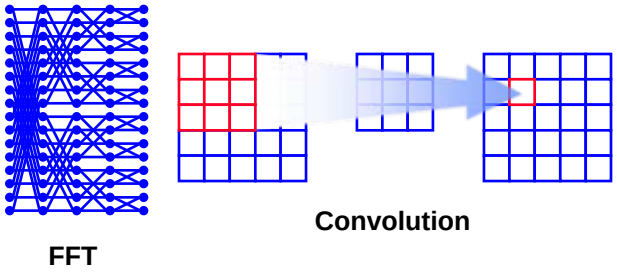
- **eXtreme Virtual Memory provides hierarchical arrays and maps**
  - **Hierarchical arrays hide details of the processor and memory hierarchy**
  - **Hierarchical maps concisely describe data distribution at each level**



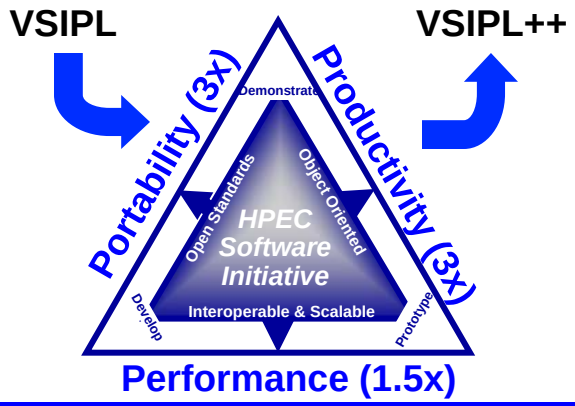
H. Kim, et al. "Advanced Hardware and Software Technologies for Ultra-long FFT's." HPEC 2005.

Computational Kernels & Processor Interface

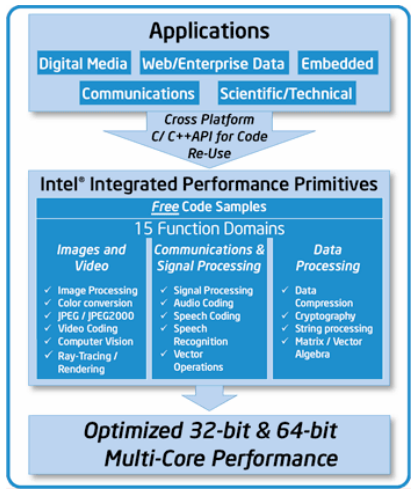
- **Mercury Scientific Algorithm Library (SAL)**



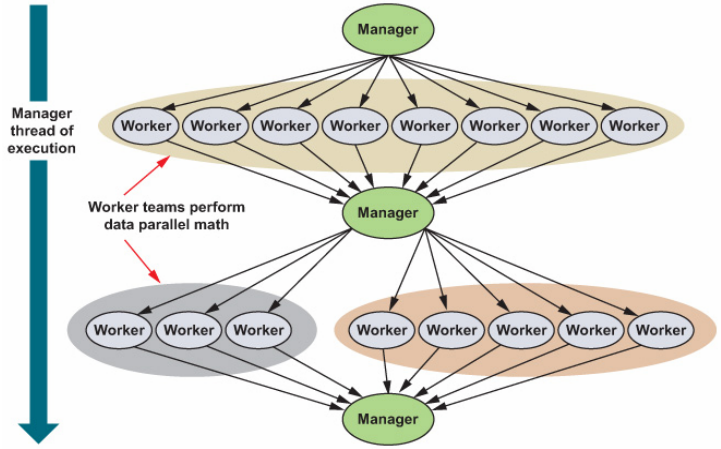
- **Vector Signal and Image Processing Library**



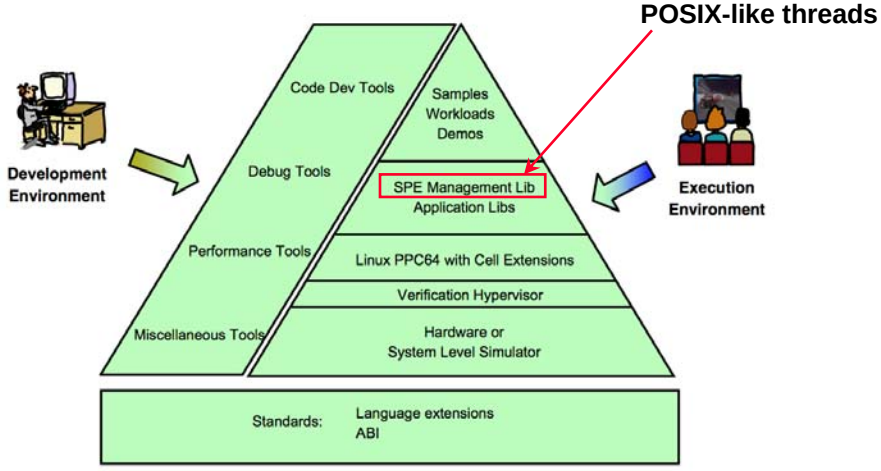
- **Intel IPP**



- **Mercury Multicore Framework (MCF)**



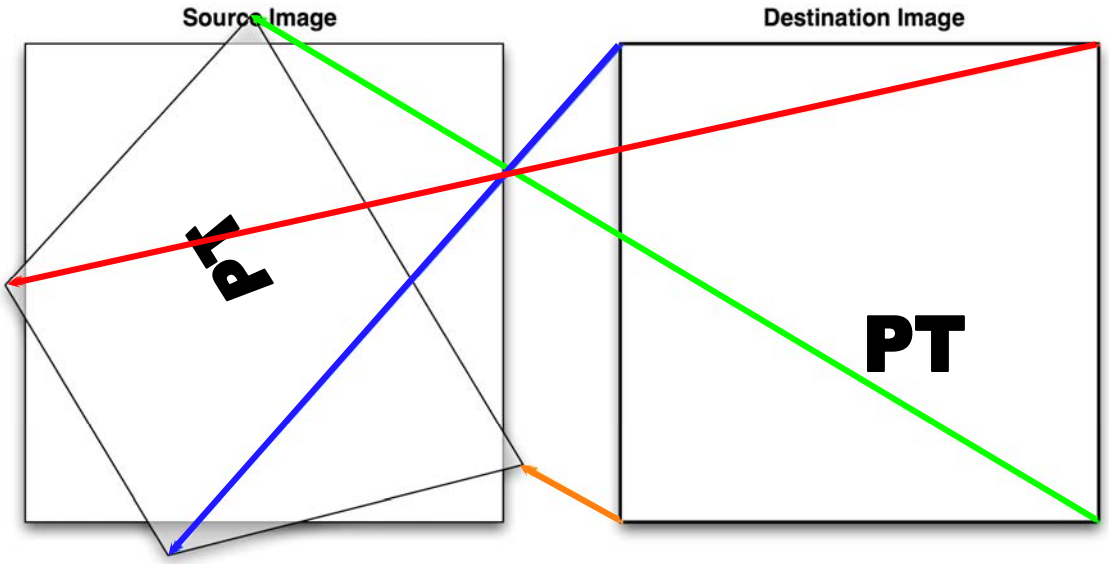
- **IBM Cell API**



Outline

- Background
- Parallel Vector Tile Optimizing Library
- Results
  - Projective Transform
  - Example Code
  - Results
- Summary

Projective Transform



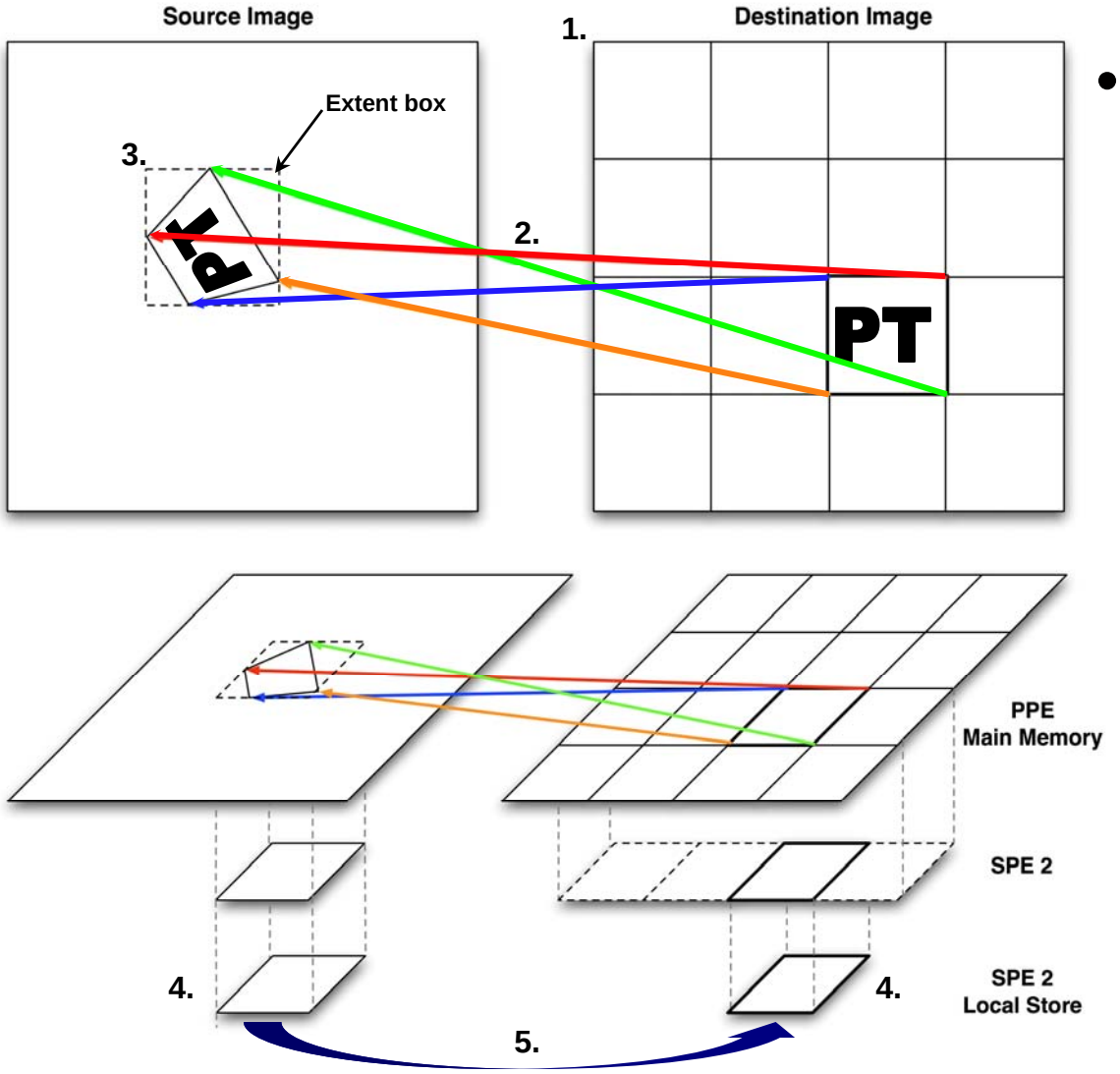
- Many DoD optical applications use mobile cameras
- Consecutive frames may be skewed relative to each other
- Standardizing the perspective allows feature extraction

- Projective transform is a homogeneous warp transform
  - Each pixel in destination image is mapped to a pixel in the source image
- Example of a real life application with a complex data distribution

S. Sacco, et al. "Projective Transform on Cell: A Case Study." HPEC 2007.

Projective Transform

Data Distribution

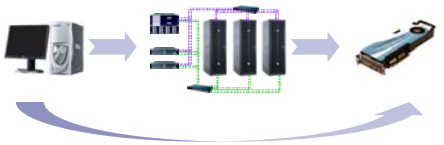


- **Mapping between source and destination pixels is data dependent**
  - **Can not use regular data distributions for both source and destination**

1. **Break destination image into blocks**
2. **Map destination block to source image**
3. **Compute extent box of source block**
4. **Transfer source and destination blocks to SPE local store**
5. **SPE applies transform to source and destination blocks**

Projective Transform Code

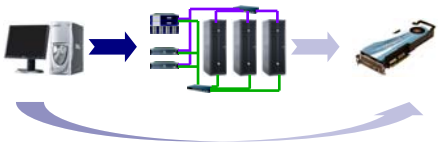
Serial



```
typedef Dense<2, short int, tuple<0, 1> > DenseBlk;
typedef Dense<2, float, tuple<0, 1> > DenseCoeffBlk;
typedef Matrix<short int, DenseBlk, LocalMap> SrcImage16;
typedef Matrix<short int, DenseBlk, LocalMap> DstImage16;
typedef Matrix<float, DenseCoeffBlk, LocalMap> Coeffs;

int main(int argc, char** argv) {
   Pvtol pvtol(argc, argv);

   // Allocate 16-bit images and warp coefficients
   SrcImage16 src(Nrows, Ncols);
   DstImage16 dst(Nrows, Ncols);
   Coeffs coeffs(3, 3);

   // Load source image and initialize warp coefficients
   loadSourceImage(&src);
   initWarpCoeffs(&coeffs);

   // Perform projective transform
   projective_transform(&src, &dst, &coeffs);
}
```

Projective Transform Code

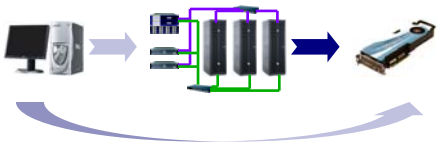
Parallel



```
typedef RuntimeMap<DataDist<BlockDist, BlockDist> > RuntimeMap;
typedef Dense<2, short int, tuple<0, 1> > DenseBlk;
typedef Dense<2, float, tuple<0, 1> > DenseCoeffBlk;
typedef Matrix<short int, DenseBlk, LocalMap> SrcImage16;
typedef Matrix<short int, DenseBlk, RuntimeMap> DstImage16;
typedef Matrix<float, DenseCoeffBlk, LocalMap> Coeffs;

int main(int argc, char** argv) {
   Pvtol pvtol(argc, argv);

   Grid dstGrid(1, 1, Grid::ARRAY); // Allocate on 1 Cell
   ProcList pList(pvtol.processorSet());
   RuntimeMap dstMap(dstGrid, pList);

   // Allocate 16-bit images and warp coefficients
   SrcImage16 src(Nrows, Ncols);
   DstImage16 dst(Nrows, Ncols, dstMap);
   Coeffs coeffs(3, 3);

   // Load source image and initialize warp coefficients
   loadSourceImage(&src);
   initWarpCoeffs(&coeffs);

   // Perform projective transform
   projective_transform(&src, &dst, &coeffs);
}
```

Projective Transform Code

Embedded



```
typedef RuntimeMap<DataDist<BlockDist, BlockDist> > RuntimeMap;
typedef Dense<2, short int, tuple<0, 1> > DenseBlk;
typedef Dense<2, float, tuple<0, 1> > DenseCoeffBlk;
typedef Matrix<short int, DenseBlk, LocalMap> SrcImage16;
typedef Matrix<short int, DenseBlk, RuntimeMap> DstImage16;
typedef Matrix<float, DenseCoeffBlk, LocalMap> Coeffs;

int main(int argc, char** argv) {
   Pvtol pvtol(argc, argv);

   // Hierarchical map for the destination image
   Grid dstTileGrid(PT_BLOCKSIZE, PT_BLOCKSIZE, Grid::ELEMENT); // Break into blocks
   DataMgmtPolicy tileDataPolicy;
   RuntimeMap dstTileMap(dstTileGrid, tileDataPolicy);

   Grid dstSPEGrid(1, pvtol.numTileProcessor(), Grid::ARRAY); // Distribute across SPE's
   ProcList speProcList(pvtol.tileProcessorSet());
   RuntimeMap dstSPEMap(dstSPEGrid, speProcList, dstTileMap);

   Grid dstGrid(1, 1, Grid::ARRAY); // Allocate on 1 Cell
   ProcList pList(pvtol.processorSet());
   RuntimeMap dstMap(dstGrid, pList, dstSPEMap);

   // Allocate 16-bit images and warp coefficients
   SrcImage16 src(Nrows, Ncols);
   DstImage16 dst(Nrows, Ncols, dstMap);
   Coeffs coeffs(3, 3);

   // Load source image and initialize warp coefficients
   loadSourceImage(&src);
   initWarpCoeffs(&coeffs);

   // Perform projective transform
   projective_transform(&src, &dst, &coeffs);
}
```

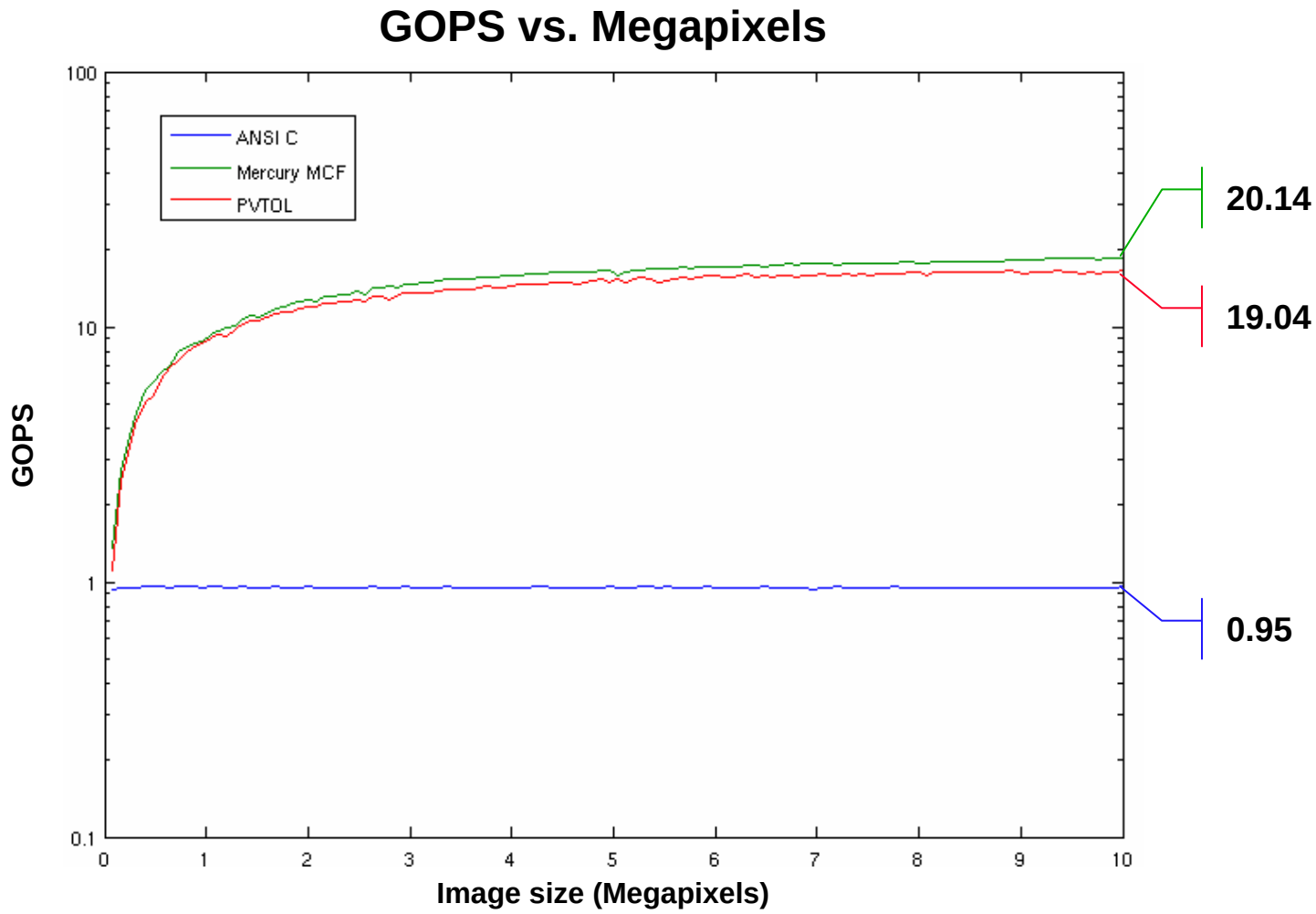
Projective Transform Code

Automapped

```
typedef Dense<2, short int, tuple<0, 1> > DenseBlk;
typedef Dense<2, float, tuple<0, 1> > DenseCoeffBlk;
typedef Matrix<short int, DenseBlk, LocalMap> SrcImage16;
typedef Matrix<short int, DenseBlk, AutoMap> DstImage16;
typedef Matrix<float, DenseCoeffBlk, LocalMap> Coeffs;

int main(int argc, char** argv) {
   Pvtol pvtol(argc, argv);

   // Allocate 16-bit images and warp coefficients
   SrcImage16 src(Nrows, Ncols);
   DstImage16 dst(Nrows, Ncols);
   Coeffs coeffs(3, 3);

   // Load source image and initialize warp coefficients
   loadSourceImage(&src);
   initWarpCoeffs(&coeffs);

   // Perform projective transform
   projective_transform(&src, &dst, &coeffs);
}
```

Results

Performance

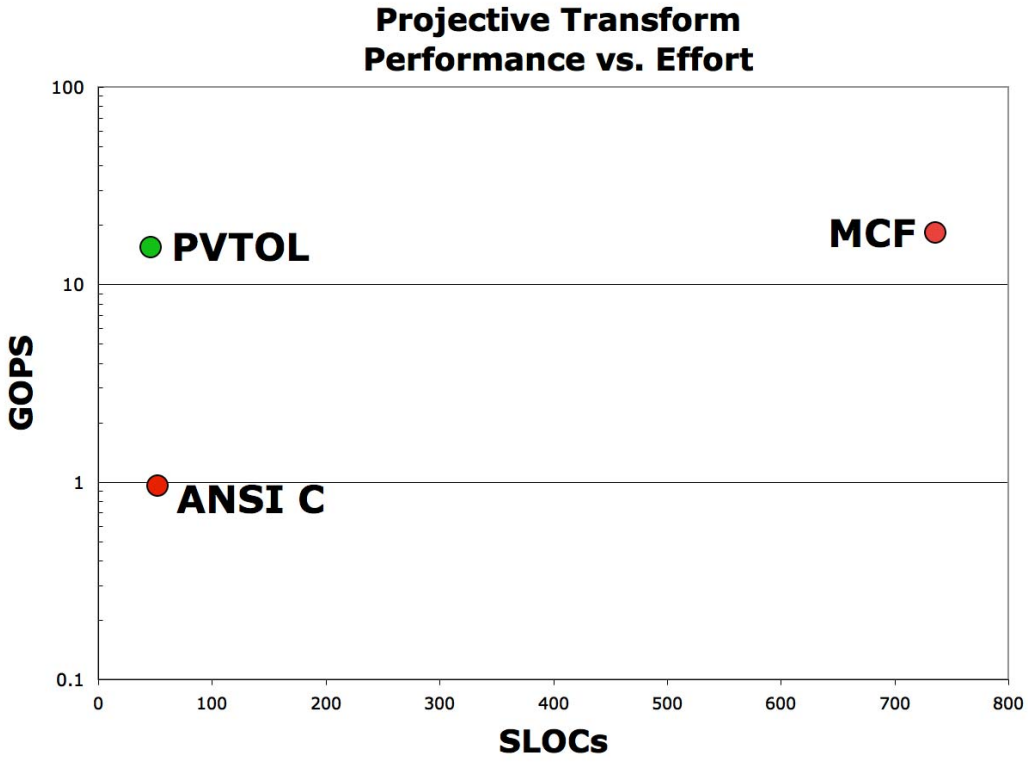


PVTOL adds minimal overhead

Results

Performance vs. Effort

	GOPS*	SLOCs
ANSI C	0.95	52
MCF	20.14	736
PVTOL	19.04	46



PVTOL acheives high performance with effort comparable to ANSI C

* 10 Megapixel image, 3.2 GHz Intel Xeon (ANSI C), 3.2 GHz Cell w/ 8 SPEs (MCF and PVTOL)

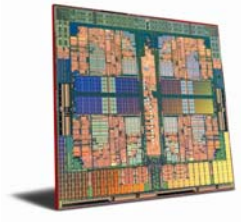
Outline

- Background
- Parallel Vector Tile Optimizing Library
- Results
- Summary

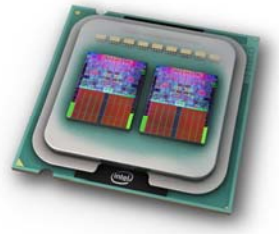
Future of Multicore Processors

- AMD vs. Intel
  - Different flavors of multicore
  - Replication vs. hierarchy

AMD Phenom

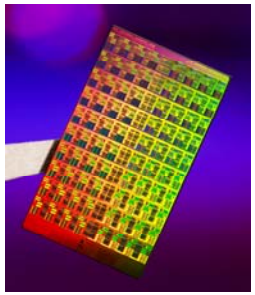


Intel Core 2 Extreme



- "Many-core" processors
  - In 2006, Intel achieved 1 TeraFLOP on an 80-core processor

Intel Polaris



- Heterogeneity
  - Multiple types of cores & architectures
  - Different threads running on different architectures

Broadcom BMC1480

GPU's

PVTOL will extend to support future multicore designs

Summary

- Emerging DoD intelligence missions will collect more data
  - Real-time performance requires significant amount of processing power
- Processor vendors are moving to multicore architectures
  - Extremely difficult to program
- PVTOL provides a simple means to program multicore processors
  - Have demonstrated for a real-life application