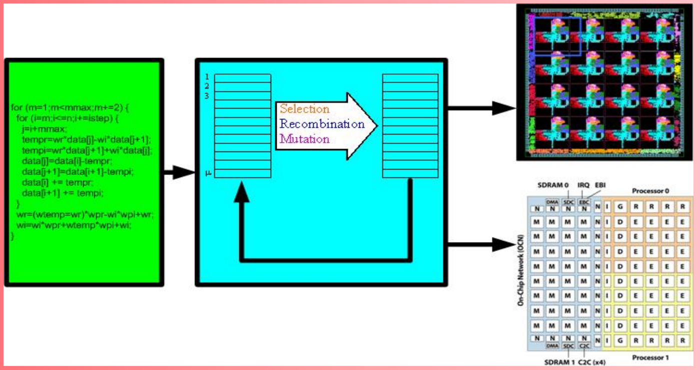


```
for (m=1;m<mmax;m+=2) {
  for (i=m;i<=n;i+=istep) {
    j=i+mmax;
    tempr=wr*data[j]-wi*data[j+1];
    tempi=wr*data[j+1]+wi*data[j];
    data[j]=data[i]-tempr;
    data[j+1]=data[i+1]-tempi;
    data[i] += tempr;
    data[i+1] += tempi;
  }
  wr=(wtemp=wr)*wpr-wi*wpi+wr;
  wi=wi*wpr+wtemp*wpi+wi;
}
```

EA-Based Generation of Compiler Heuristics for Polymorphous Computing Architectures

Laurence D. Merkle, Michael C. McClurg,
Matt G. Ellis, Tyler G. Hicks-Wright

Department of Computer Science and Software Engineering
Rose-Hulman Institute of Technology
5500 Wabash Ave, CM-103, Terre Haute, IN
merkle@rose-hulman.edu

This work was supported by the Advanced Computing Architectures Branch, Information Directorate, Air Force Research Laboratory, Air Force Materiel Command, USAF, under grant FA8750-05-1-0019.

MSAEC – July 8, 2006– Seattle, WA

Outline

- Research Objective
- PCA Background
  - DARPA PCA Program
  - MIT RAW Project
  - UT Austin TRIPS Project
- Technical Progress
  - Parallel EAs for PCAs
  - Taxonomy of EAs in Compilers
  - EA-Generation of TRIPS Compiler Heuristics
- Future Directions

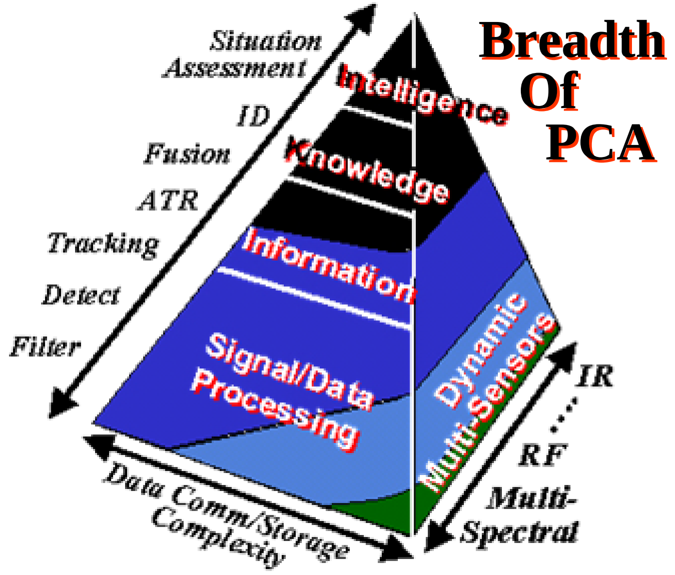
Research Objective

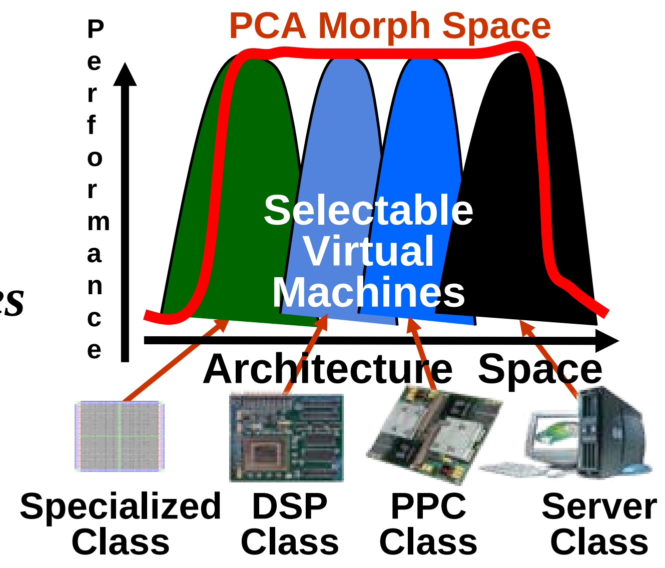
- Enhance Raw and TRIPS compilers to produce more efficient executable code
- Approach:
  - Adopt optimization view of compilation
  - Hybridize EC techniques with existing algorithms

PCA Objectives

GOAL

- **Processing Diversity**
  1. Breadth of processing ⟶ *Four Classes*
     1. Signal/Data Processing
     2. Information
     3. Knowledge
     4. Intelligence
  2. Uniform performance ⟶ *Competitive with best-in-class*
- **Mission Agility**
  1. In mission (retargetable) ⟶ *msec*
  2. Mission – to - mission ⟶ *Days*
  3. New threat scenario ⟶ *Months*
- **Architectural Flexibility**
  1. High efficiency selectable virtual machines ⟶ *Two+ Major Morph States*
  2. Mission Adaptation (Constraints e.g. energy) ⟶ *N Minor Morph States*
  3. Portability/Evolvability ⟶ *Two Layer Morphware*





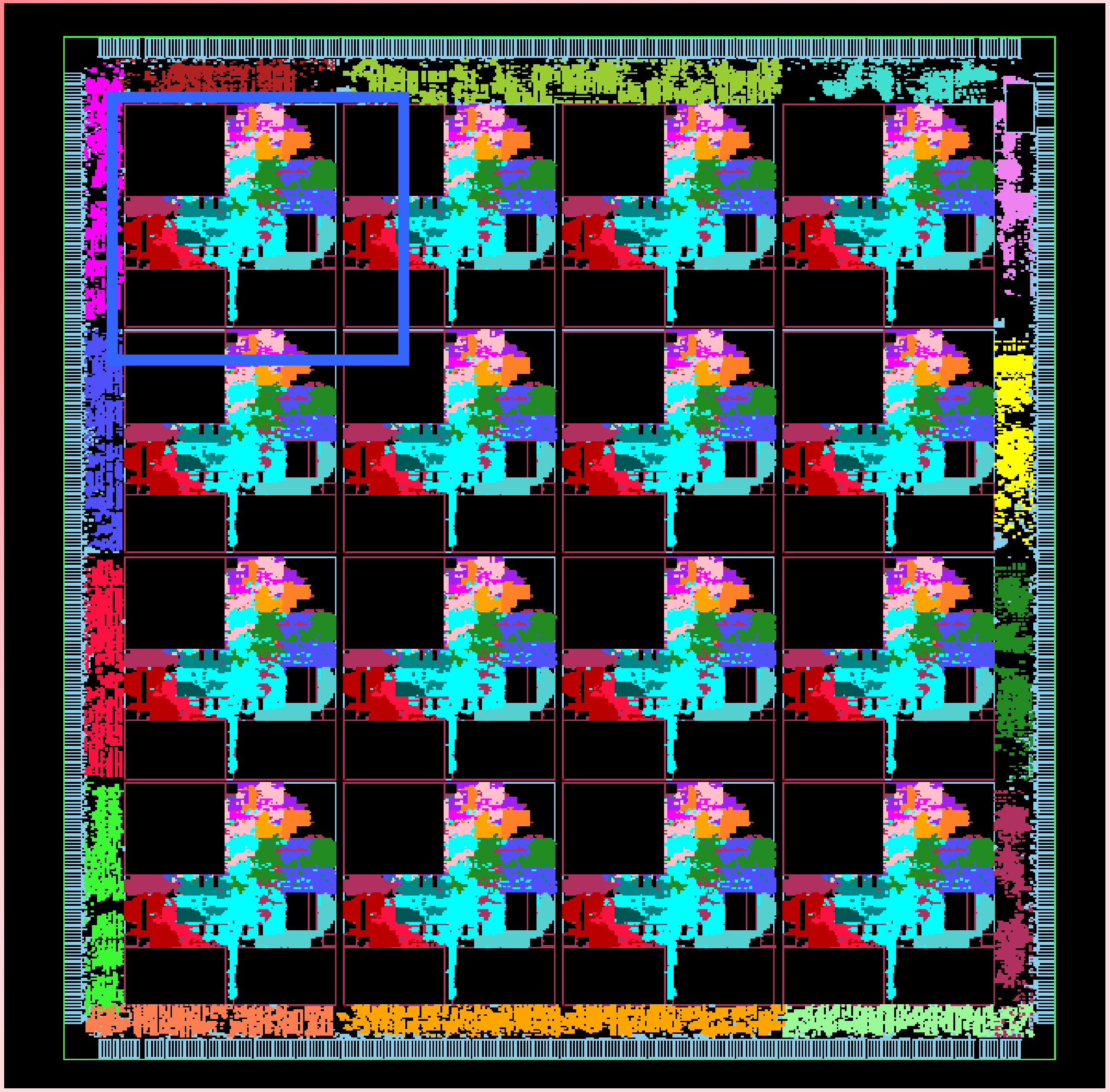
PCA technology supports agile processing as a function of dynamic mission objectives and constraints

Tile-based Architectures

Context:

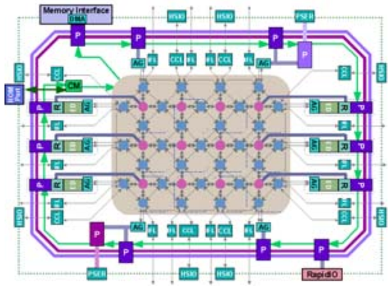
- Ongoing improvements in manufacturing technology
- Wire delays becoming more significant relative to gate delays

Leading PCA efforts achieve dynamic responsiveness and scalability through use of tile-based architectures

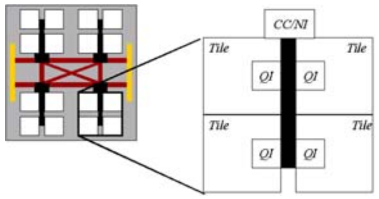


PCA Hardware Overview

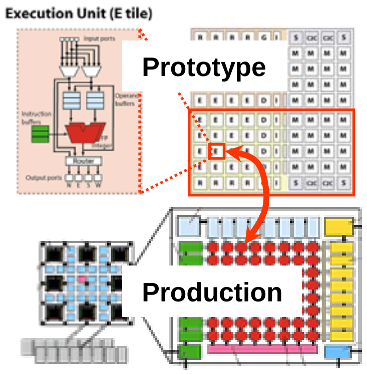
- ISI/Raytheon/Mercury: MONARCH/MCHIP



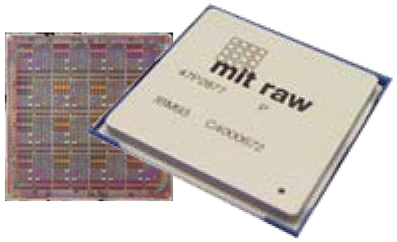
- Stanford: Smart Memories



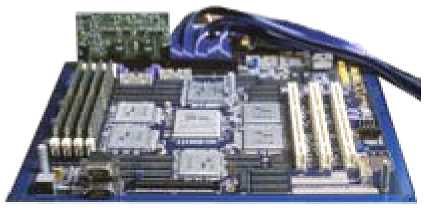
- University of Texas/IBM: TRIPS



- MIT: RAW (Early Prototype)



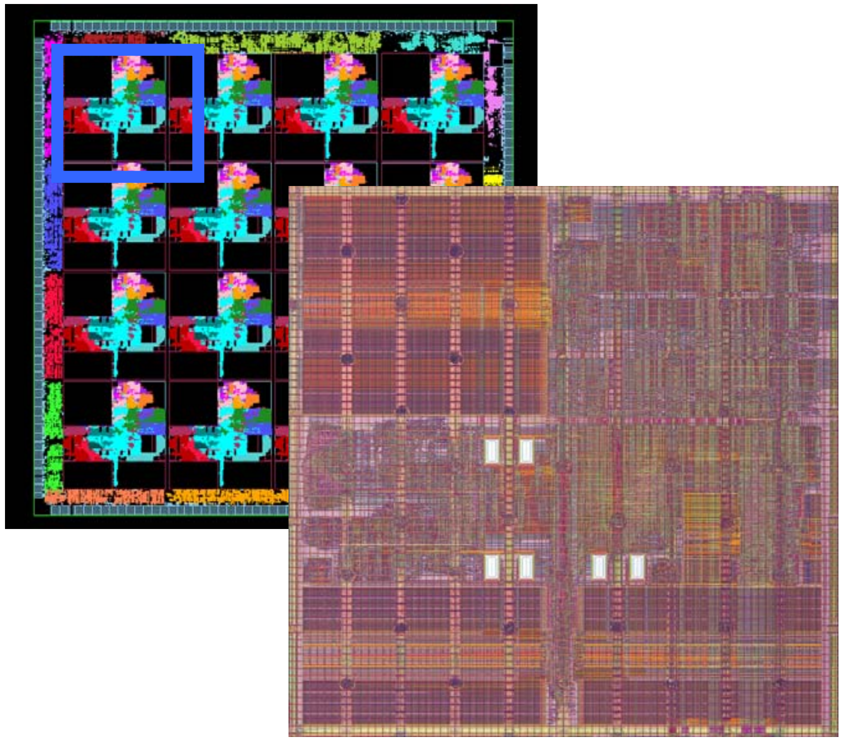
- MIT/LL: Early Testbed

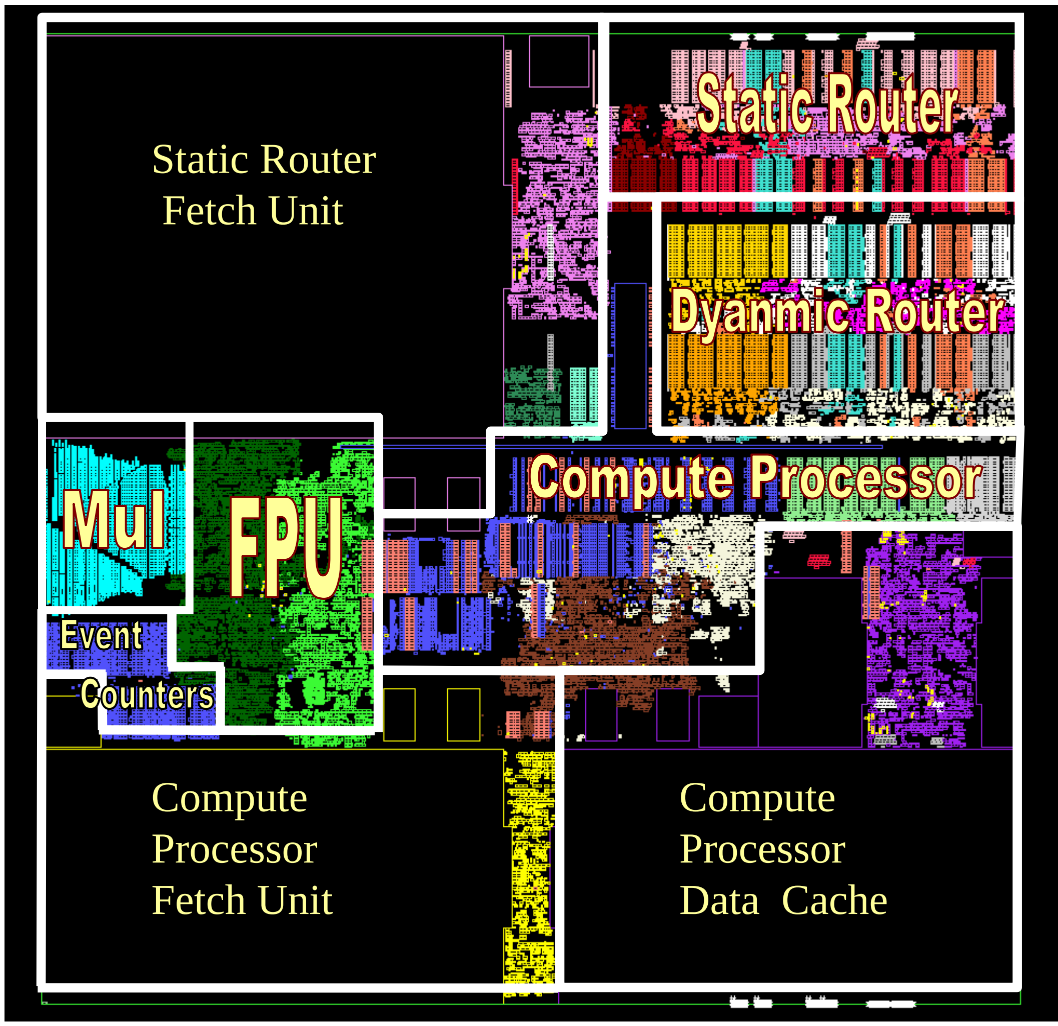


RAW Architecture (Agarwal, et al.)

- Raw Architecture Workstation (RAW) fully exposes low-level details of architecture to compiler
  - Allows compiler or software to optimize resource allocation for each application
- Compiler generates traditional machine instructions and “switch instructions” for each tile
- Two orders of magnitude better performance than traditional processors in simulations of certain applications

The Raw Architecture



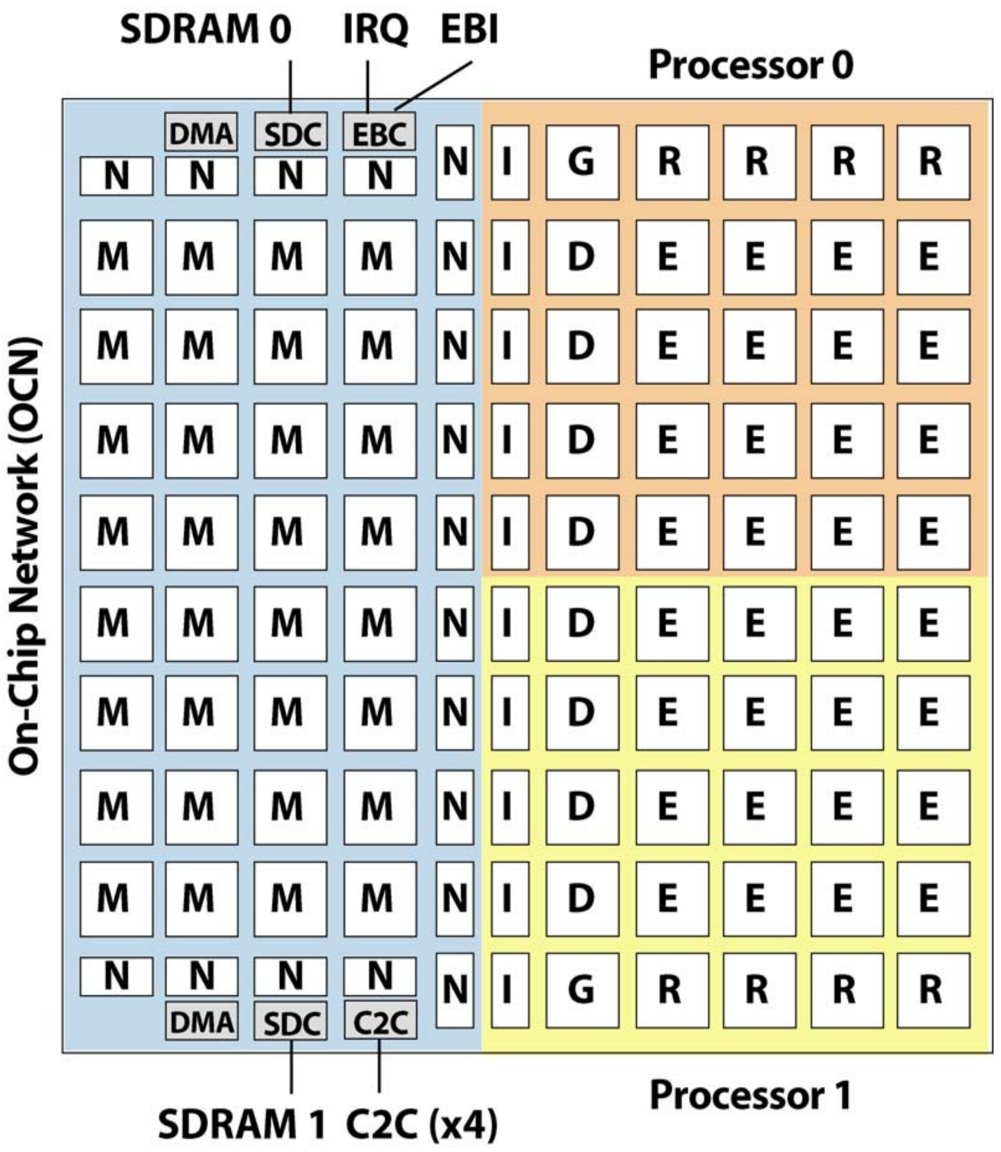


- Divide the silicon into an array of identical, programmable tiles
  - A signal can get through a small amount of logic and to the next tile in one cycle
- Tiles connected by software-exposed on-chip interconnect
  - Scalar Operand Network [HPCA 03]

TRIPS Architecture (Burger, et al.)

- Grid Processor Architectures
  - Composed of tightly coupled array of ALUs connected by thin network
  - Producer instruction outputs delivered directly as consumer instruction inputs
    - Example of Explicit Data Graph Execution (EDGE) Architecture
- Tera-op Reliable and Intelligently Adaptive Processing System (TRIPS)
  - One or more grid processors working in parallel
  - Sensor network monitors application behavior, feeds back to runtime system, application, and compiler

TRIPS Chip Floorplan



TRIPS Tiles and Interfaces

G:	Processor control - dispatch, next block predictor, commit
R:	Register file - 32 registers x 4 threads, register forwarding
I:	Instruction cache - 16KB, 16-entry TLB variable-size pages
D:	Data cache - 8KB, 64-entry load/store queue, 16-entry TLB
E:	Execution unit - 128 reservation stations, integer/FP ALUs
M:	Memory - 64KB, OCN router with 4 virtual channels
N:	OCN network interface - OCN router, PA translation
DMA:	Direct memory access controller
SDC:	SDRAM controller
EBC:	External bus controller (to PowerPC)
C2C:	Chip-to-chip network links - to four neighbors
IRQ:	Interrupt request - service request to PowerPC
EBI:	External bus interfaces - command interface from PPC

Compiling for the TRIPS Architecture (Burger, et al.)

- Difficult multicriteria optimization problem.
  - Compiler must be able to
    - Identify basic blocks
    - Partition basic blocks into hyperblocks
    - Map hyperblocks to tiles
    - Map each operation to an execution unit
  - Spatial scheduling affects both concurrency and communications delays
- Compiler currently employs a greedy approximation algorithm

Technical Progress: Parallel EAs for PCAs

- Enabling steps
  - Toolchains obtained from developers
  - Correct installations verified
- Parallel evolutionary algorithms
  - Island-model implementations designed and implemented for both architectures
  - Empirical evaluations in progress

Technical Progress: Taxonomy of EAs in Compilers

- Identified and described opportunities for additional compiler optimizations
- General classification of methods for application of EC to compilation
- Raw – no automatic scheduling to optimize
  - Programmer must partition source code
  - EC no more and no less applicable to Raw than to any MIPS compiler
- TRIPS – compiler partitions instructions into hyperblocks

Classification of Methods for Application of EC to Compilation

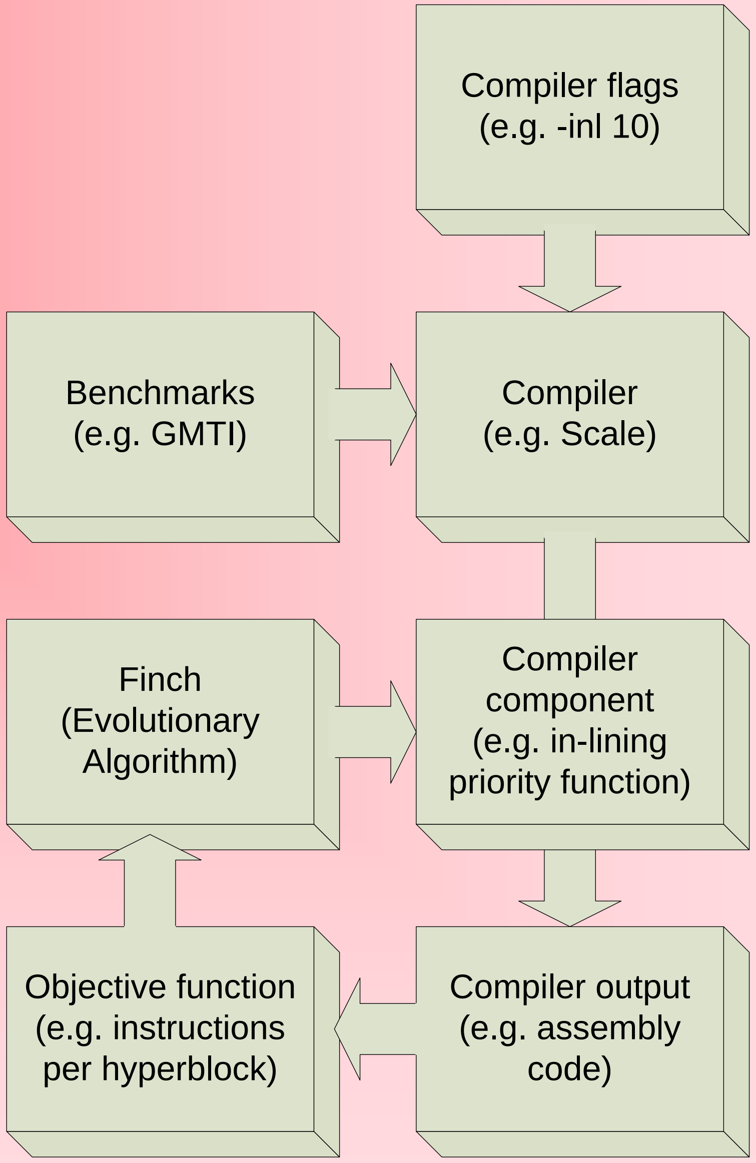
	Effectiveness – Application execution time	Efficiency – Compiler execution time	Reproducibility (w/o controlling random number generator)
Compiler Algorithms	Unconstrained search for best overall compiler	Can be explicitly considered as an objective function	Compilation
Compiler Parameters	Regular and nearly unconstrained search space	Can be explicitly considered as an objective function	Compilation
Compile Time	Unconstrained search for fastest executable	Too slow for use in development environment	Execution
Schedule Time	Dynamic search for fastest execution	Essentially unchanged	None

Technical Progress: EA-Generation of TRIPS Compiler Optimization Heuristics

- EA-based generation of more effective TRIPS compiler heuristics through integration of
    - Finch Meta-Optimization Framework
    - TRIPS C compiler
- Metrics
    - Hyperblock count
    - Instructions per hyperblock
    - Ratio of hyperblocks to instructions per hyperblock
    - Static vs. dynamic
    - SPEC2000 Benchmarks
    - PCA Community Benchmarks

Finch Meta-optimization Framework (Stephenson, et al.)

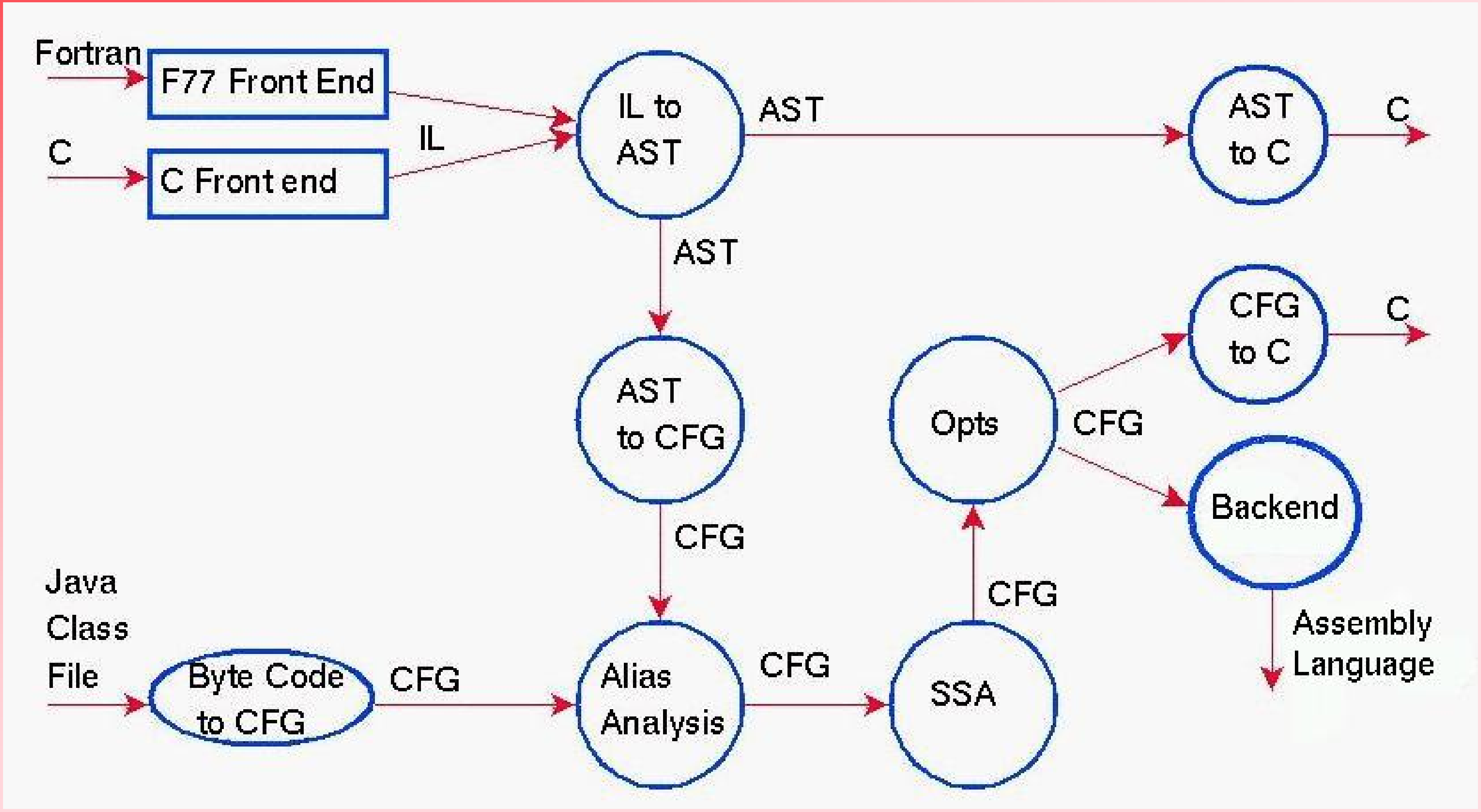
- Compiler heuristic to be optimized replaced by indirect invocation of EA-provided heuristic
- EA searches space of heuristics that have the required signature
- Candidate heuristics are evaluated by invoking compiler, then using objective function to evaluate compiler output
- EA executes on master processor, compiler and objective function execute on slave processors


Compiler flags (e.g. -inl 10)
Benchmarks (e.g. GMTI)
Compiler (e.g. Scale)
Finch (Evolutionary Algorithm)
Compiler component (e.g. in-lining priority function)
Objective function (e.g. instructions per hyperblock)
Compiler output (e.g. assembly code)

TRIPS C Compiler

- Scalable Compiler for Analytical Experiments (Scale)
  - Developed by Department of Computer Science, University of Massachusetts, Amherst
- Implemented by shell script wrappers around Scale compiler
- Modular compiler, easily modified for different output platforms
- Emphasis on hyperblock formation for TRIPS and other platforms

Scale Compiler Data Flow Diagram



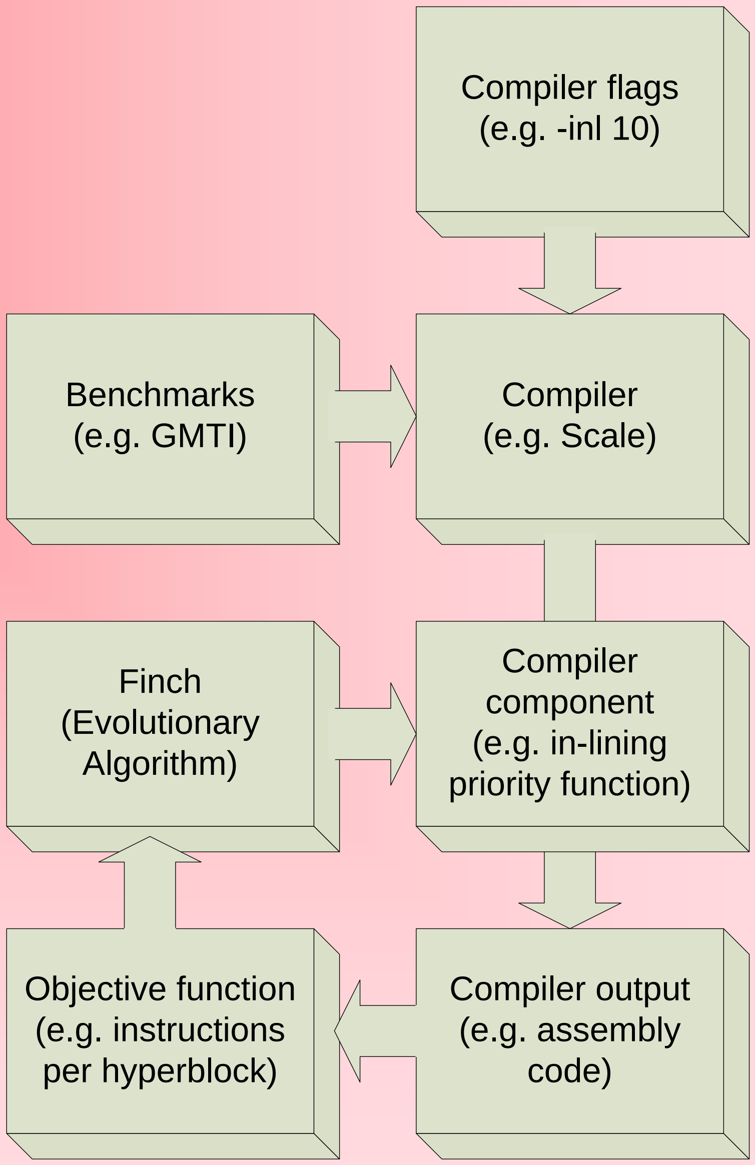
Example Code

```
int main(int argc, char** argv) { printf("Hello world\n"); return 0; }
```

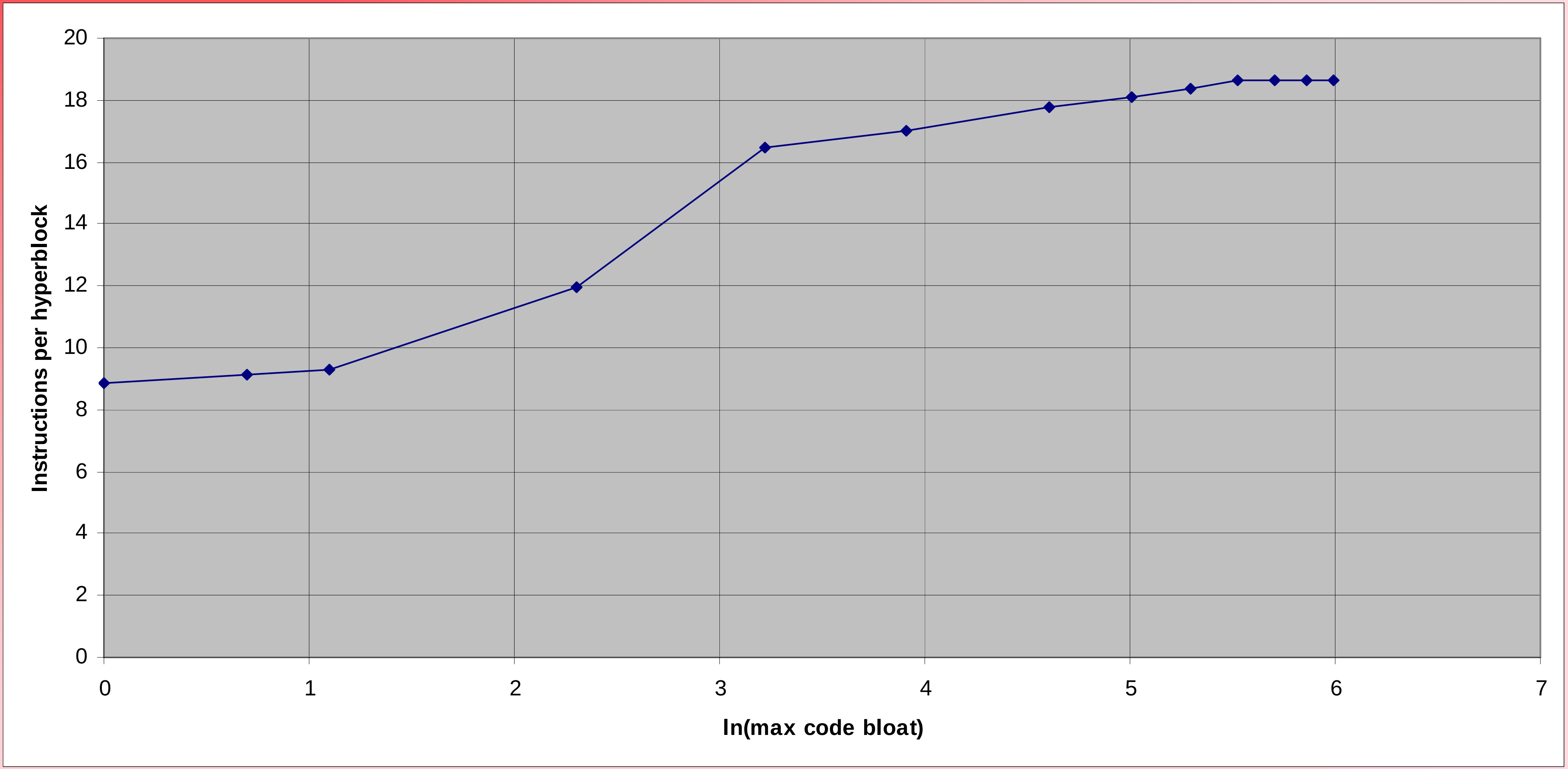
```
.app-file "test.tcc.c"
.data
.align 8
_V2:
      .ascii     "Hello world\n\000"

.text
.global main
.bbegin main
      read       $t0, $g1
      read       $t1, $g2
      addi       $t2, $t0, -96
      sd         -88($t0), $t1 S[0]
      entera     $t3, _V2
      enterb     $t4, main$1
      callo      printf
      addi       $t2, $t0, -96
      sd         -88($t0), $t1 S[0]
      entera     $t3, _V2
      enterb     $t4, main$1
.     callo      printf
      addi       $t2, $t0, -96
      sd         -88($t0), $t1 S[0]
      entera     $t3, _V2
      enterb     $t4, main$1
      callo      printf
      write      $g1, $t2
      write      $g2, $t4
      write      $g3, $t3
.bend
.bbegin main$1
      read       $t0, $g1
      movi       $t1, 0
      ld         $t2, 8($t0) L[0]
      addi       $t3, $t0, 96
      ret        $t2
      write      $g1, $t3
      write      $g2, $t2
      write      $g3, $t1
.bend
```

Application of Finch to Scale

- Ported Scale to Finch environment
- Converted Finch to MPI-based communication
- Identified candidate heuristic functions
  - (Re-)construction of abstract syntax tree
  - Construction of control flow graph
  - Standard post-translation optimizations
  - In-lining priorit


Compiler flags (e.g. -inl 10)
Benchmarks (e.g. GMTI)
Compiler (e.g. Scale)
Finch (Evolutionary Algorithm)
Compiler component (e.g. in-lining priority function)
Objective function (e.g. instructions per hyperblock)
Compiler output (e.g. assembly code)

In-Lining Technique Results: GMTI Benchmark



References

- Agarwal, A. Raw Computation. Scientific American, August, 1999.
- Burger, D., S. Keckler, K. McKinley, M. Dahlin, L. John, C. Lin, C. Moore, J. Burrill, R. McDonald, W. Yoder, and the TRIPS Team. "Scaling to the End of Silicon with EDGE Architectures." IEEE Computer, July 2004, pp. 44-55.
- Goldberg, D. The Design of Innovation. Kluwer Academic Publishers, Boston, 2002.
- Stephenson, et al. "Meta Optimization: Improving Compiler Heuristics with Machine Learning". MIT.
- Scale project website (http://osl-www.cs.umass.edu/Scale), Scale Compiler Group, Department of Computer Science, University of Massachusetts, Amherst.
- Taylor, M., J. Kim, J. Miller, D. Wentzlaff, F. Ghodrat, B. Greenwald, H. Hoffmann, P. Johnson, J.-W. Lee, W. Lee, A. Ma, A. Saraf, M. Seneski, N. Shnidman, V. Strumpen M. Frank, S. Amarasinghe and A. Agarwal. The Raw Microprocessor: A Computational Fabric for Software Circuits and General Purpose Programs. IEEE Micro, Mar/Apr 2002.