PICO: ASIC Synthesis from C

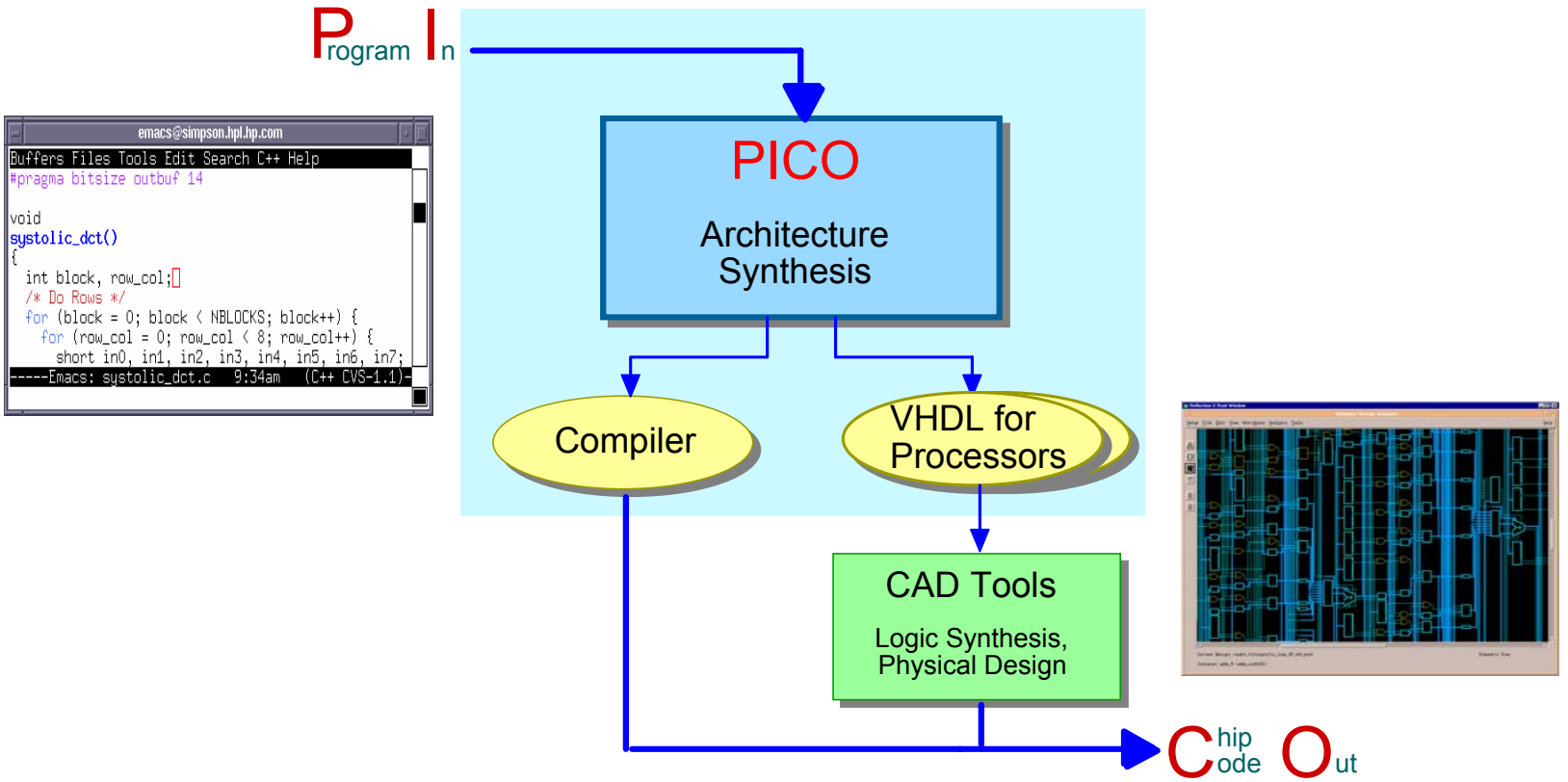
Rob Schreiber Shail Aditya Bob Rau
Vinod Kathail Scott Mahlke
Darren Cronquist Mukund Sivaraman

HP Labs, Palo Alto

Outline

- What Can PICO Do for an SOC Designer?
- The PICO System Design Hierarchy
- From Sequential to Parallel Loop Nest
- Parallel Loop Nest to Processor Design

PICO overview

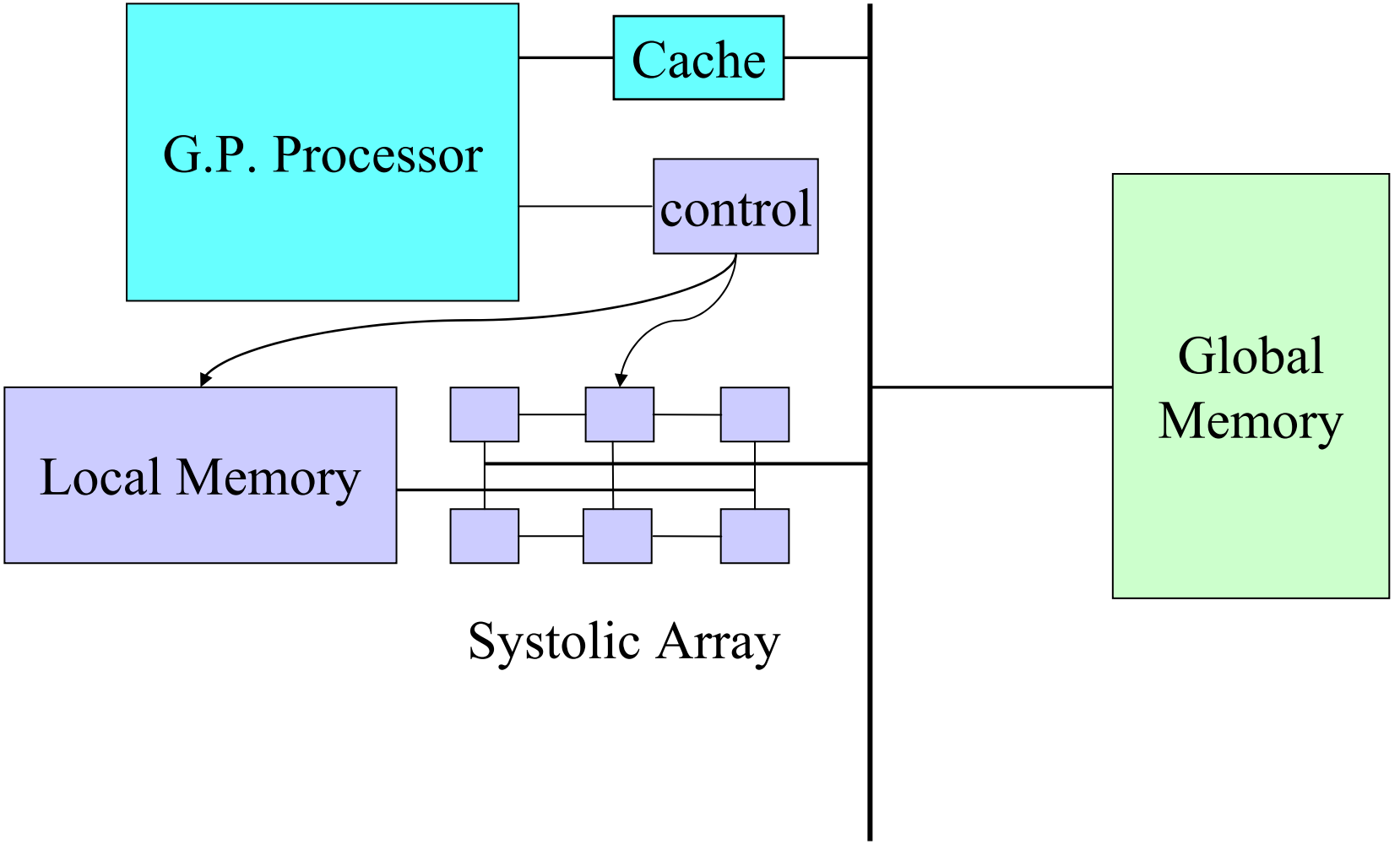


Program In --> IP Out

Using PICO

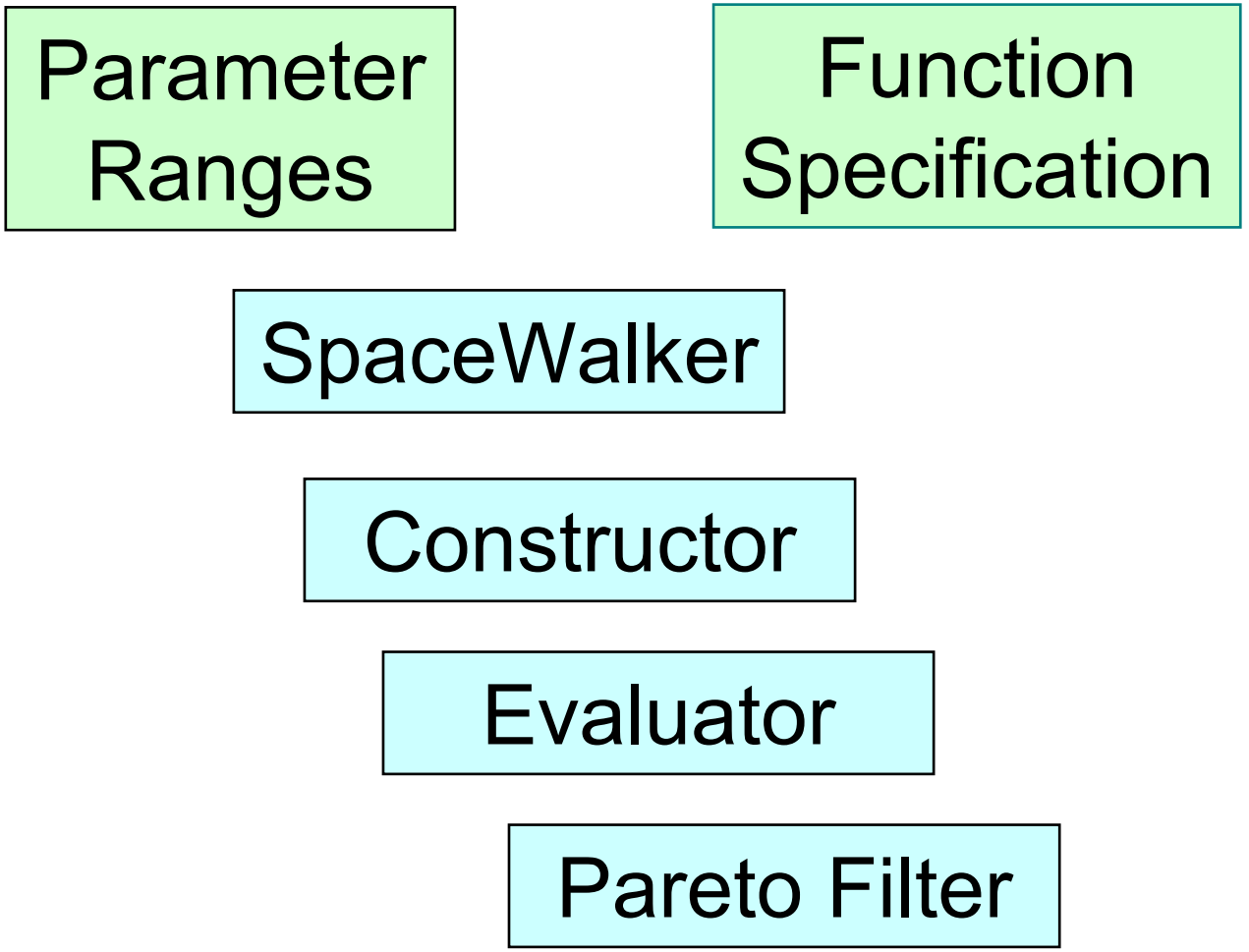
- User provides application, test data, and design space limits
- User indicates hot loop nests
- PICO creates Pareto set of ASIP designs.
- Each design has a customized VLIW with zero or more loop nests realized in HW
- User selects appropriate design for SOC based on area, power, performance tradeoff

PICO's ASIP Architecture



Hierarchical Design Frameworks

An Automated Design Template

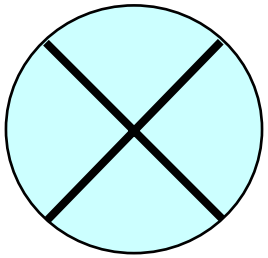


Good Systems from Good Subsystems

Cache Pareto

VLIW Pareto

NPA Pareto

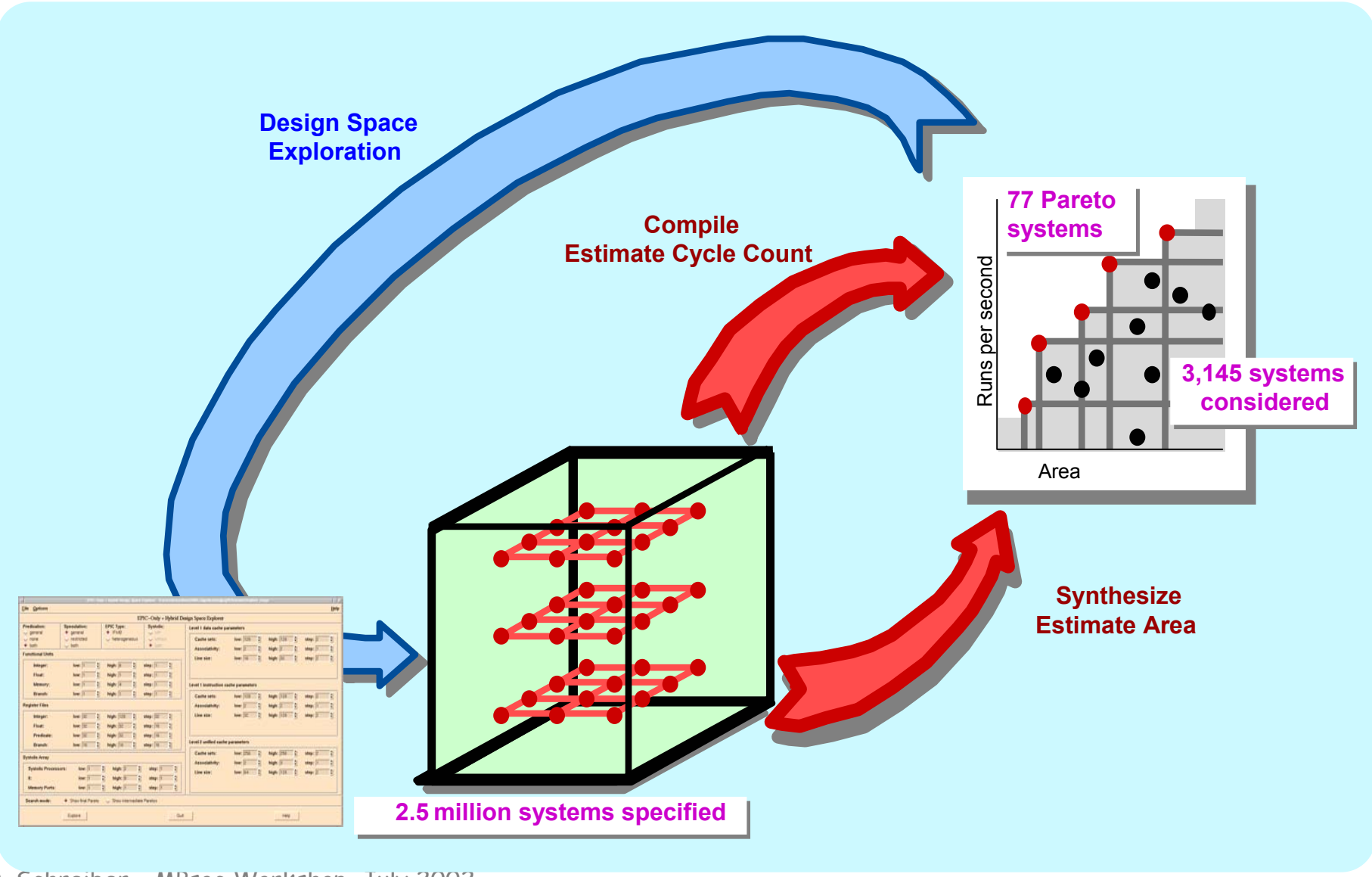


System Constructor

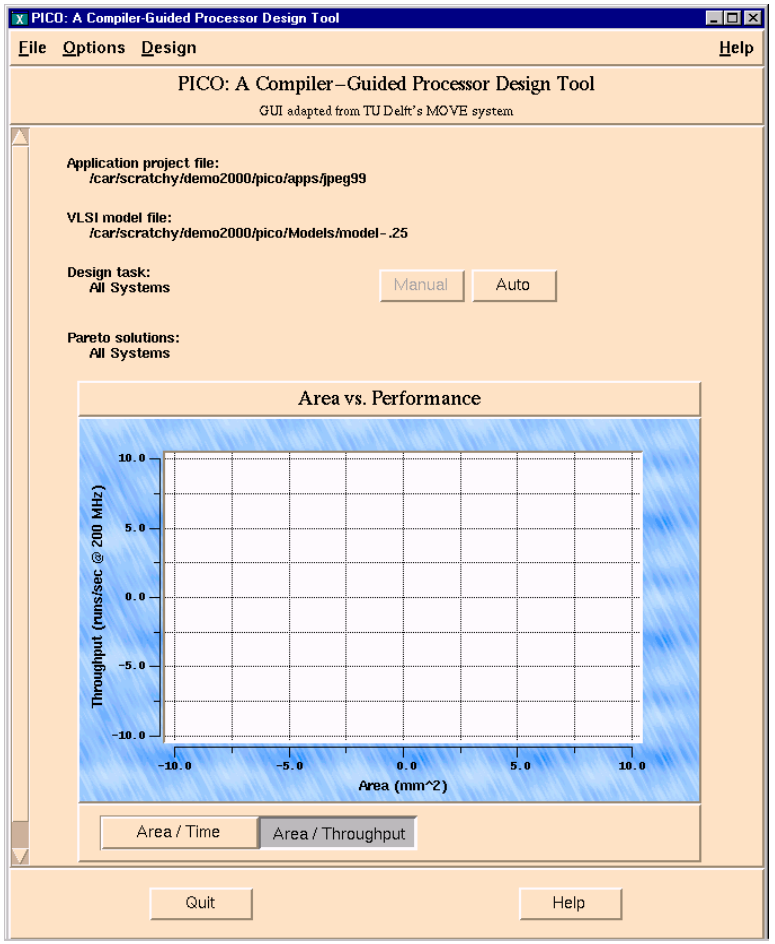
System Evaluator

System Pareto Filter

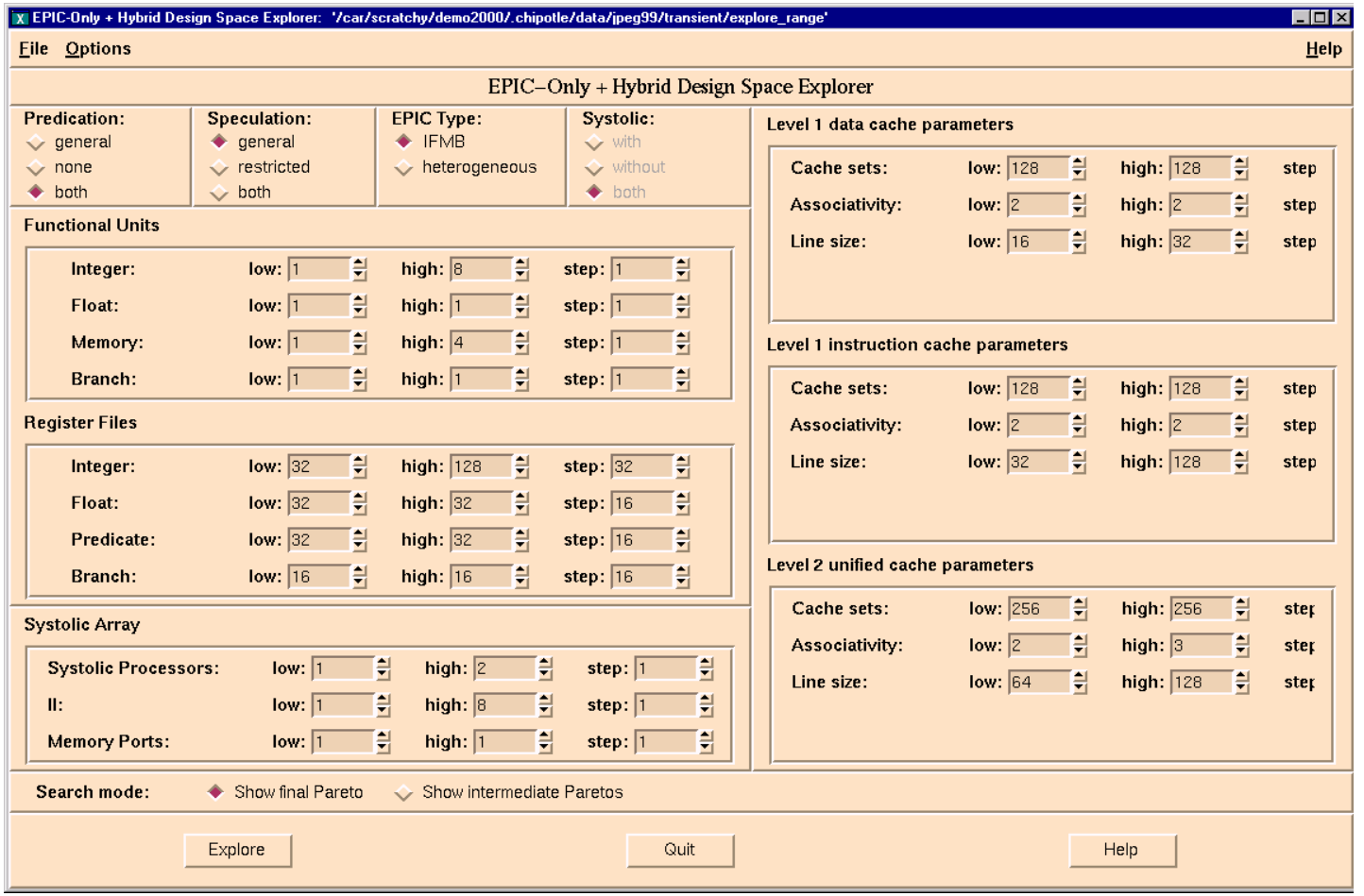
design space exploration



PICO GUI

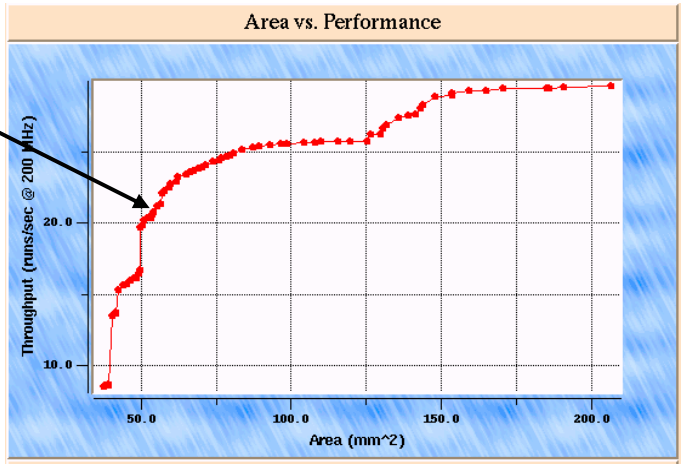

PICO: A Compiler-Guided Processor Design Tool
File
Options
Design
Help
PICO: A Compiler–Guided Processor Design Tool
GUI adapted from TU Delft's MOVE system
Application project file:
/car/scratchy/demo2000/pico/apps/jpeg99
VLSI model file:
/car/scratchy/demo2000/pico/Models/model-.25
Design task:
All Systems
Manual
Auto
Pareto solutions:
All Systems
Area vs. Performance
Throughput (runs/sec @ 200 MHz)
10.0
5.0
0.0
-5.0
-10.0
-10.0
-5.0
0.0
5.0
10.0
Area (mm^2)
Area / Time
Area / Throughput
Quit
Help

Limiting the Design Space

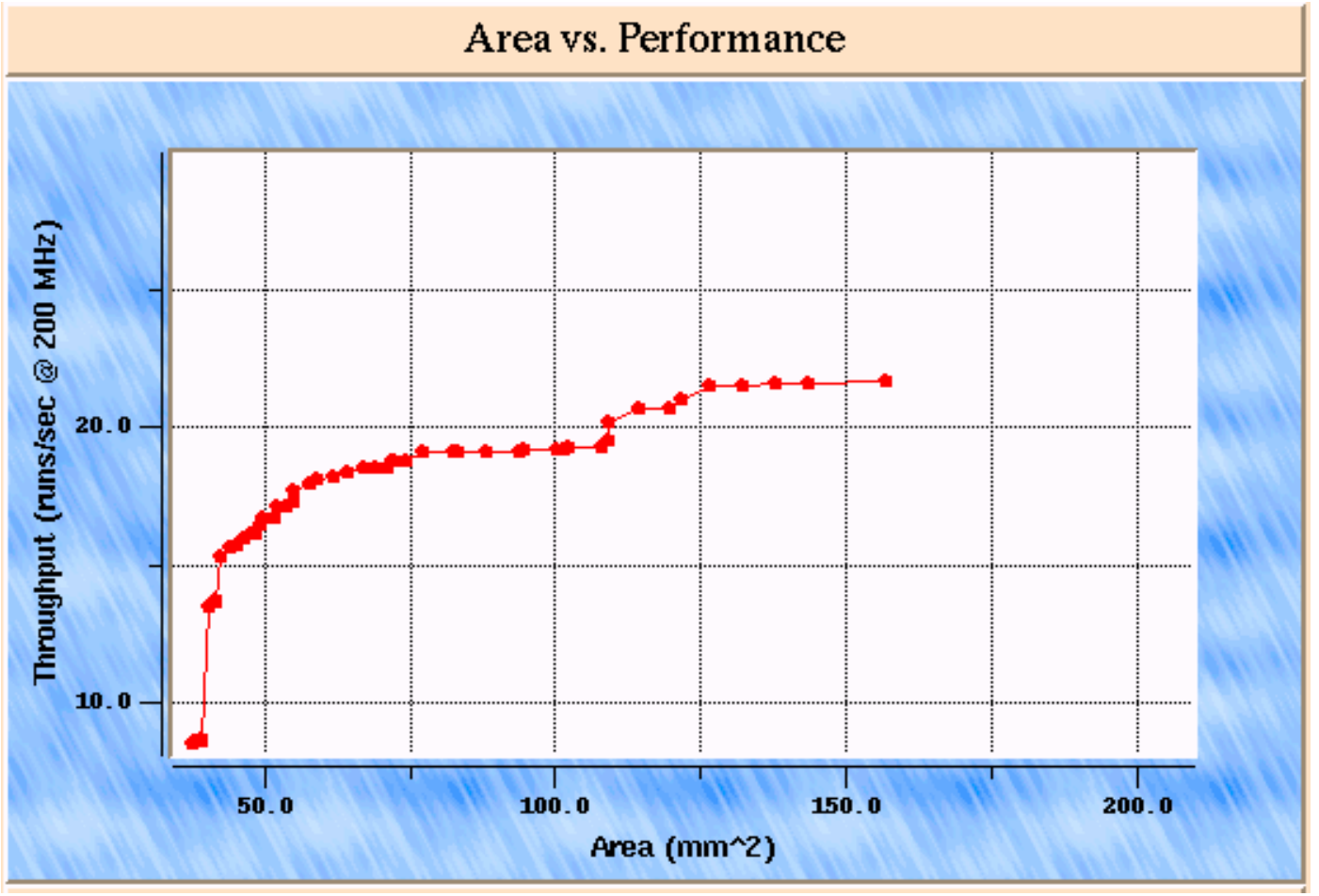


Exploration

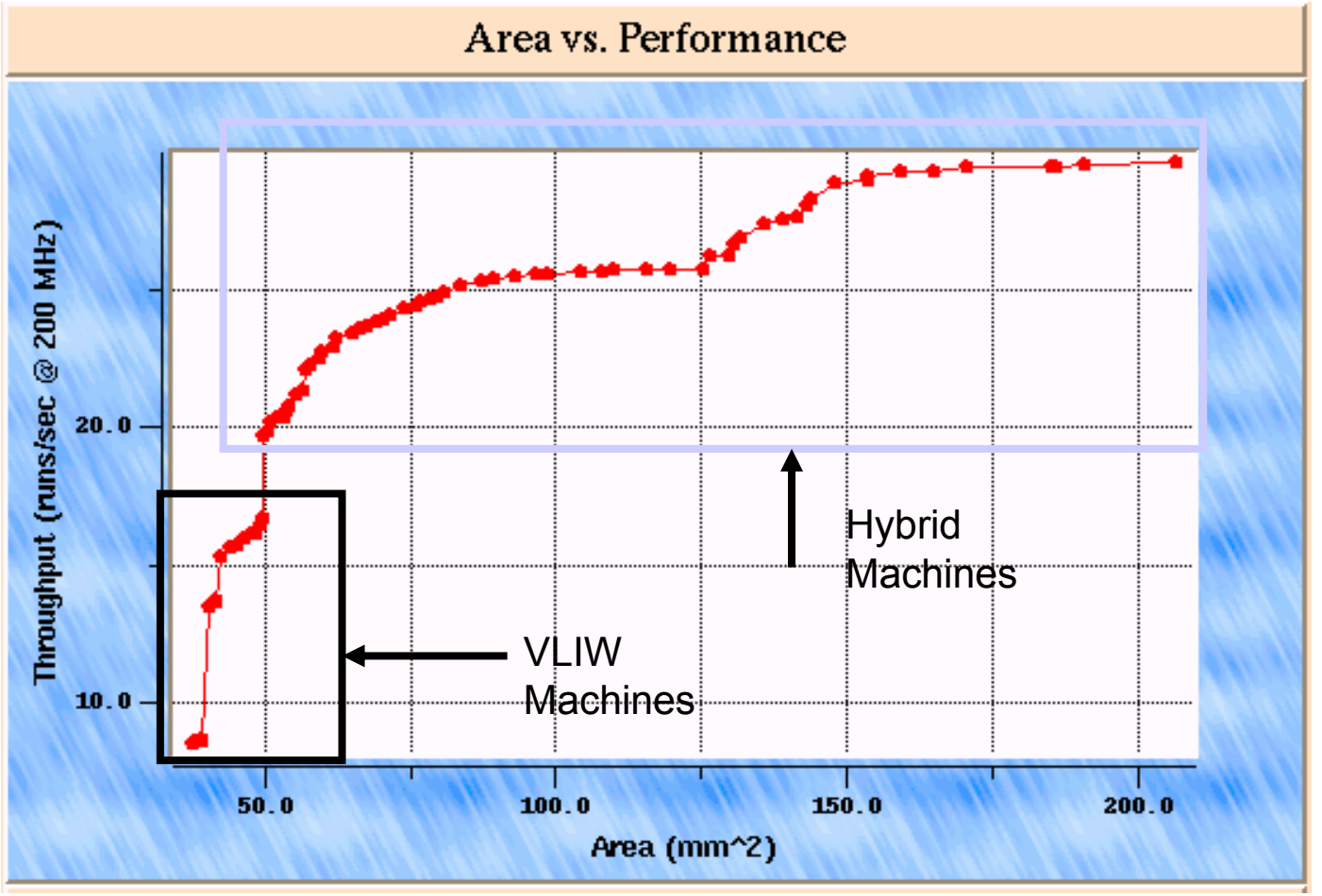
```
hpterm (simpson via TELNET)
>>    [systemHandle = 15861, expHandle = 15871]
>>    area = 68.5671, cycles = 9.81964e+06, packetSize = 512, textSize = 105624 (0.828125)
>> Evaluated jpeg99 on vliw(p0s1 i6f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15881, expHandle = 15891]
>>    area = 72.2403, cycles = 9.8348e+06, packetSize = 512, textSize = 98728 (0.828125)
>> Added vliw(p1s1 i7f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT) 15901
>> Added vliw(p0s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT) 15921
>> Added vliw(p0s1 i7f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT) 15941
>> STEP 11: 66 machines evaluated
>> Selected vliw(p1s1 i7f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT)
>> Selected vliw(p0s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT)
>> Selected vliw(p0s1 i7f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT)
>> Evaluated jpeg99 on vliw(p1s1 i7f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15901, expHandle = 15911]
>>    area = 70.946, cycles = 1.05966e+07, packetSize = 512, textSize = 132284 (1)
>> Evaluated jpeg99 on vliw(p0s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15921, expHandle = 15931]
>>    area = 75.6306, cycles = 8.92249e+06, packetSize = 512, textSize = 115676 (1)
>> Evaluated jpeg99 on vliw(p0s1 i7f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15941, expHandle = 15951]
>>    area = 79.9293, cycles = 9.81246e+06, packetSize = 512, textSize = 106200 (0.828125)
>> Added vliw(p1s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT) 15961
>> Added vliw(p0s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT) 15981
>> STEP 12: 69 machines evaluated
>> Selected vliw(p1s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT)
>> Selected vliw(p0s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT)
>> Evaluated jpeg99 on vliw(p1s1 i8f1m4b1 i[64,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15961, expHandle = 15971]
>>    area = 77.2607, cycles = 1.05966e+07, packetSize = 512, textSize = 141764 (1.19531)
>> Evaluated jpeg99 on vliw(p0s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 15981, expHandle = 15991]
>>    area = 88.8775, cycles = 8.91315e+06, packetSize = 512, textSize = 112832 (1)
>> Added vliw(p1s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT) 16001
>> Added vliw(p0s1 i8f1m4b1 i[128,0]f[32,0]p[32,0]b[16] IFMT) 16021
>> STEP 13: 71 machines evaluated
>> Selected vliw(p1s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT)
>> Selected vliw(p0s1 i8f1m4b1 i[128,0]f[32,0]p[32,0]b[16] IFMT)
>> Evaluated jpeg99 on vliw(p1s1 i8f1m4b1 i[96,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 16001, expHandle = 16011]
>>    area = 90.7583, cycles = 1.059e+07, packetSize = 512, textSize = 144724 (1.19531)
>> Evaluated jpeg99 on vliw(p0s1 i8f1m4b1 i[128,0]f[32,0]p[32,0]b[16] IFMT):
>>    [systemHandle = 16021, expHandle = 16031]
>>    area = 100.992, cycles = 8.92426e+06, packetSize = 512, textSize = 111312 (1)
>> STEP 14: 73 machines evaluated
>> STEP 15: 73 machines evaluated
>> Composing (vliw +  memory) paretos.....................................
>> done.
>> Refining system cost estimates...
>> done.
```



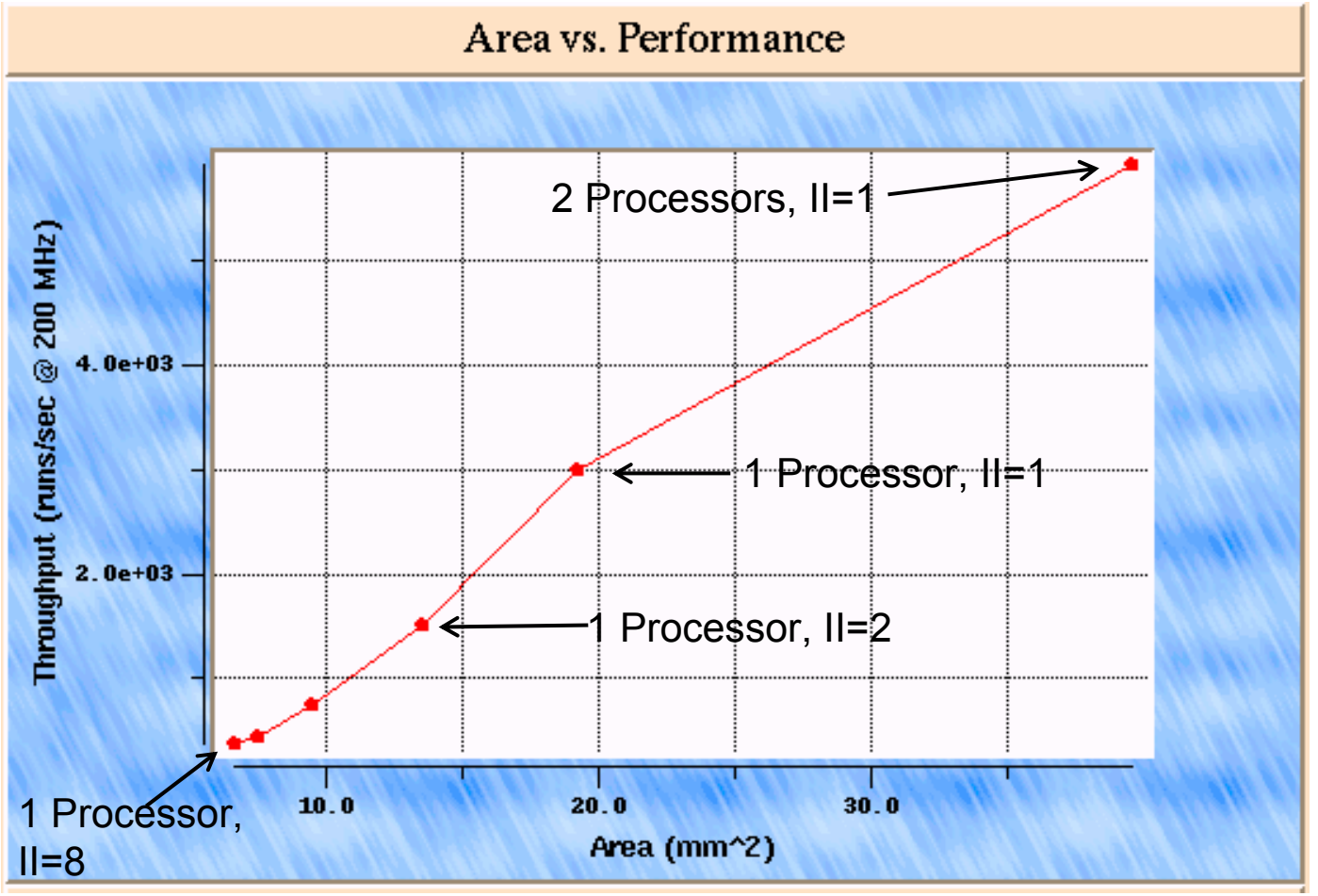
Pareto Optimal Machines: VLIW-only



Pareto Optimal Machines: All systems



Systolic Design: Exploration



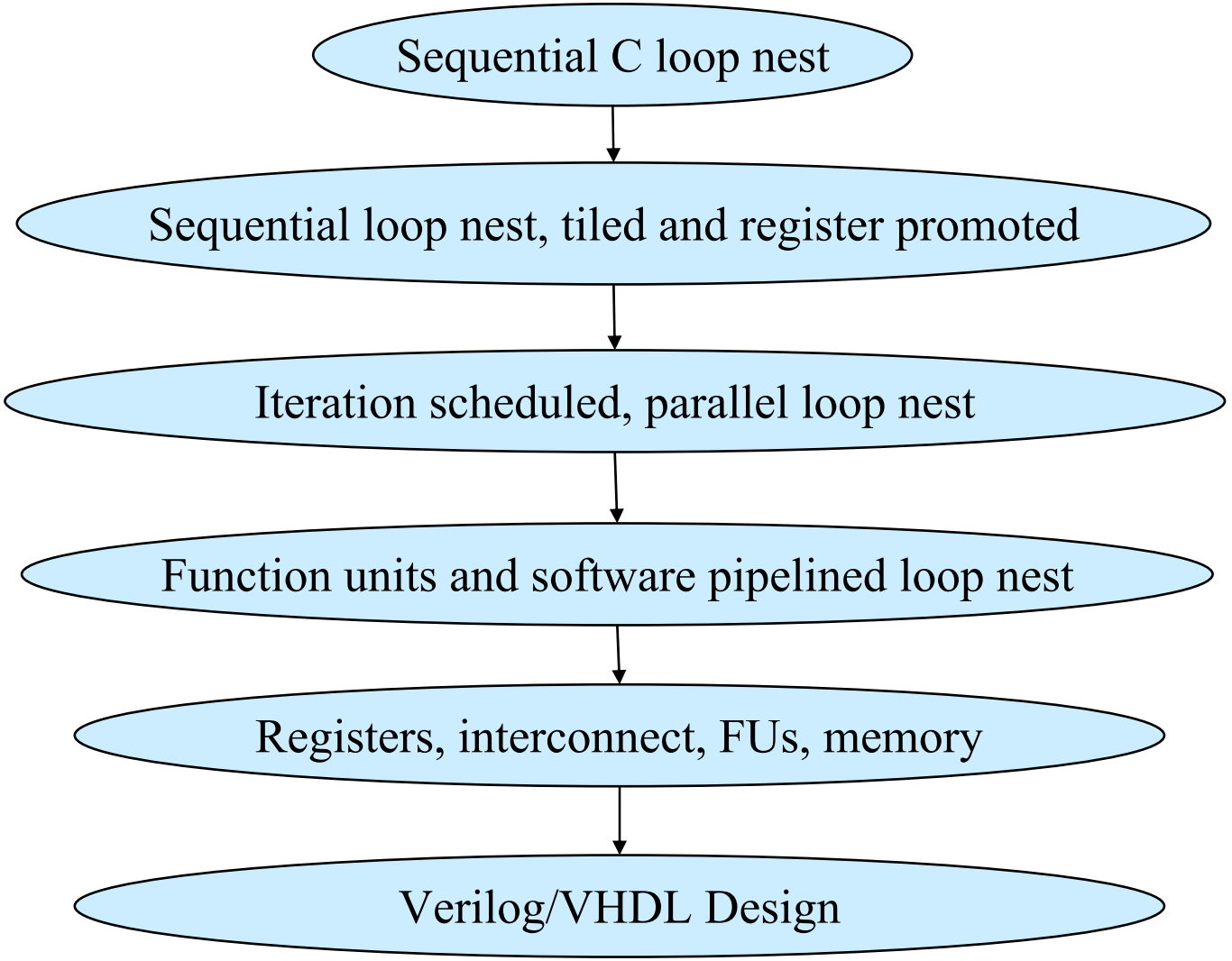
Synthesis of a Non-Programmable, Application-Specific Accelerator:

From Sequential Loop Nest to Parallel Loop Nest

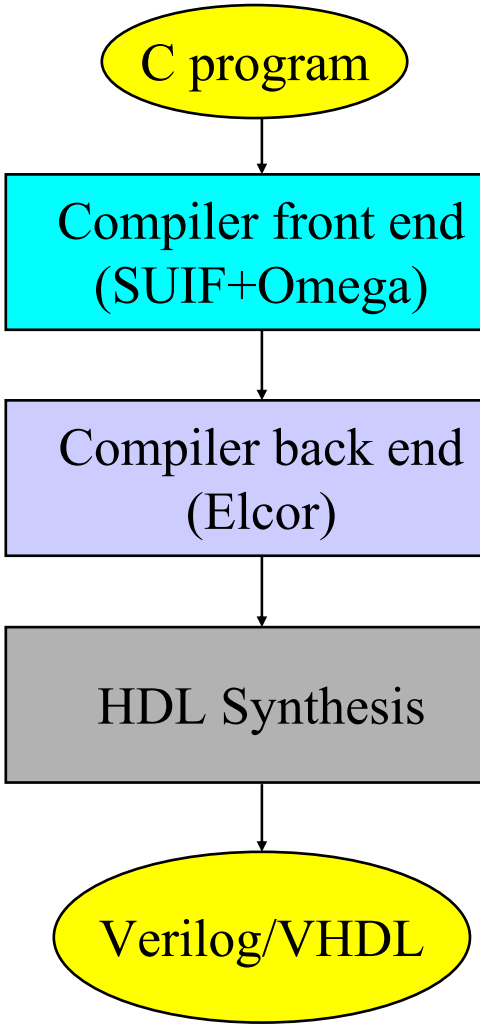
Input Language

- A perfect loop nest → A systolic array
- A sequence of nests → A pipeline of arrays
- Constant loop bounds
- Dependence analysis must be feasible:
  - No aliasing through pointers
- Language extensions
    - `#pragma bitsize x 12`
    - `#internal coeff`

From C to VHDL



From C to VHDL



Tiles, schedules, maps, transforms loops, eliminates loads/stores

Optimizes, analyzes bitwidth, allocates function units, software pipelining

Allocates registers and interconnect. Builds VHDL description of processor.

What does it take to make this efficient?

The Memory Wall

CPU

Memory

Cache and Local Memory

CPU

Cache

DSP/NPA

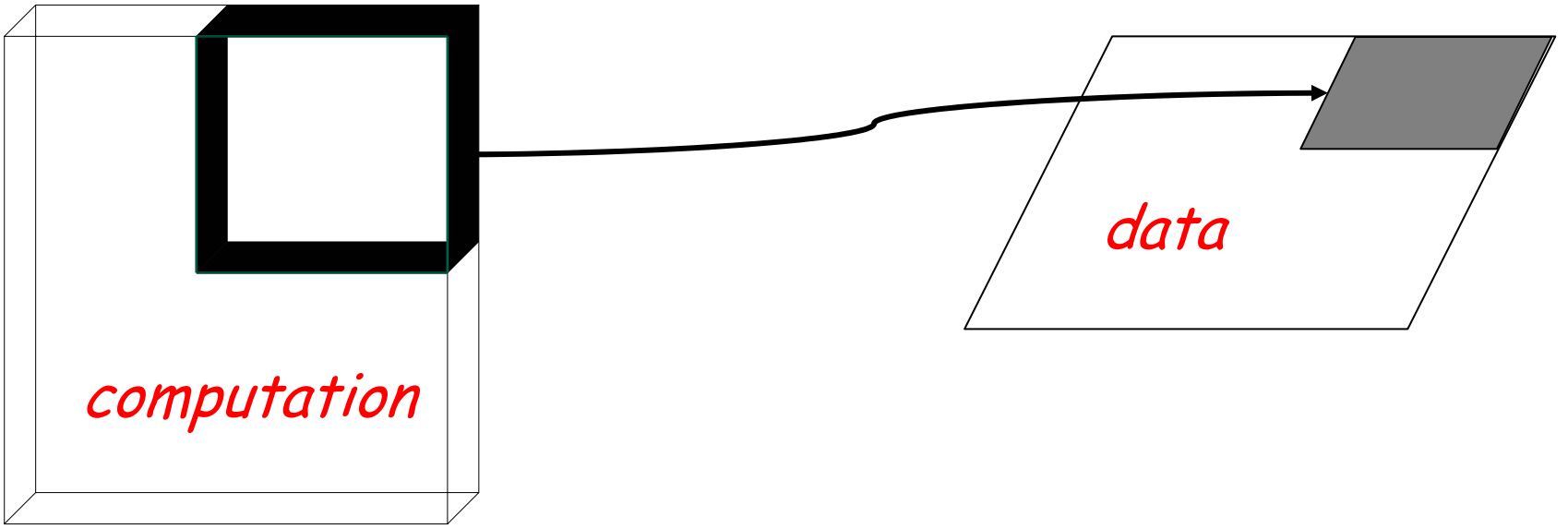
Local Memory

Memory

Goal of Code Transformation

```
for each TILE
 {
  for (t = 0; t < Tfinal; t++)
   {
    forall processors p
     {
       X[t][p] = . . .
          Y[t-1][p+1] . . .
     }
   }
 }
```

Tiling the Iteration Space



Volume/Surface = O(radius)

Computation/Footprint = Ω(radius)

Computation/Footprint = CPU/Memory

Load/Store Elimination

- For affine array references, intermediate results in registers
- For affine, read-only array references, data routed through registers; no value loaded more than once.

Tile Shapes

Big tiles → More local memory

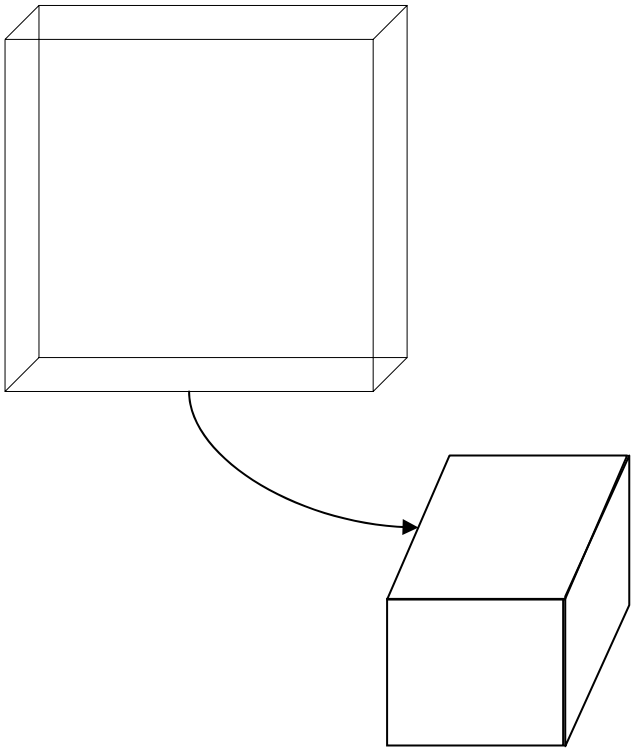
Small tiles → less reuse of data, more global memory bandwidth

Optimal tile → smallest tile that does not oversubscribe memory bandwidth

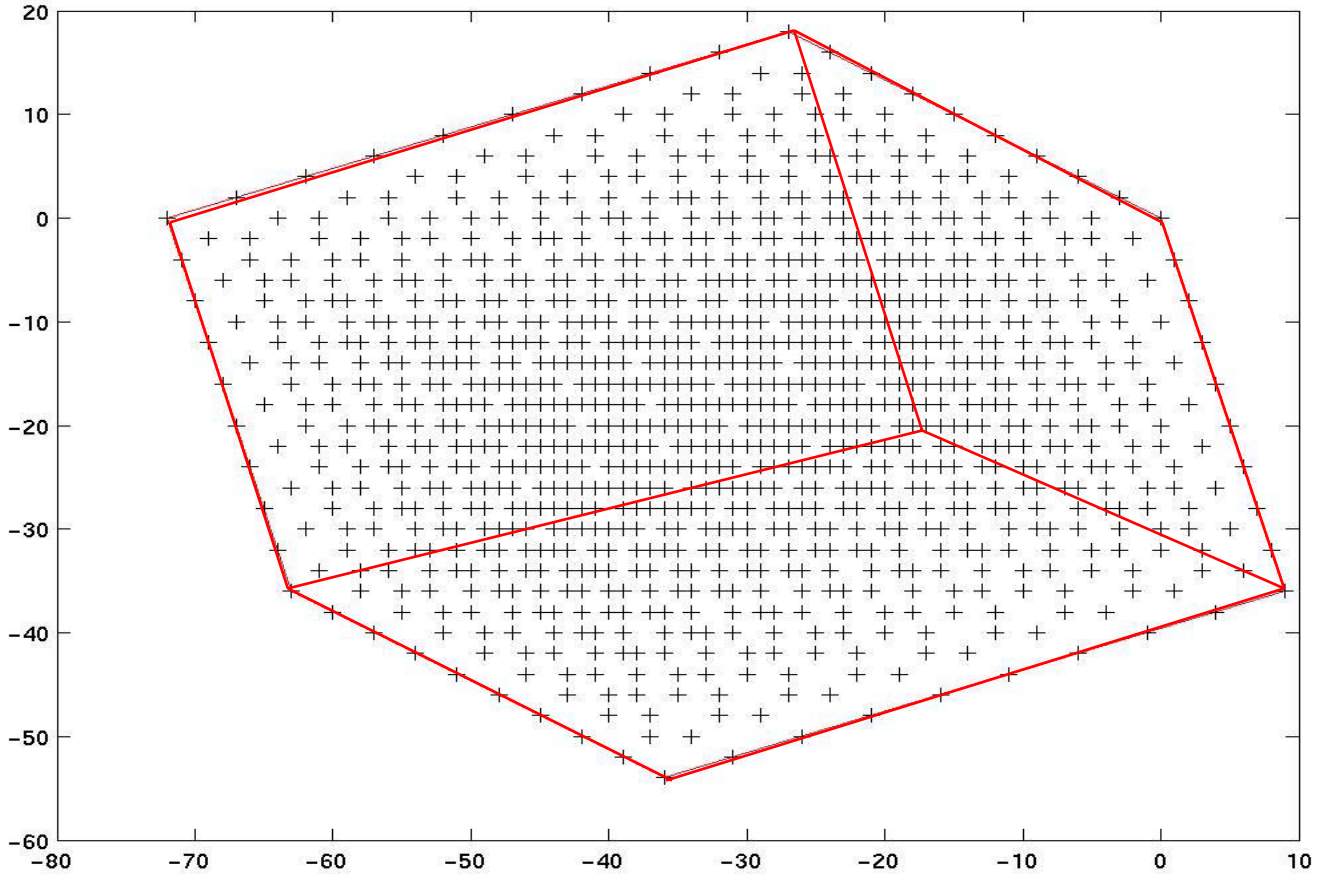
Estimating the Footprint

Affine array reference
X[i+j][2*j-3*k]

How many integer points in an affine image of a rectangular iteration space?



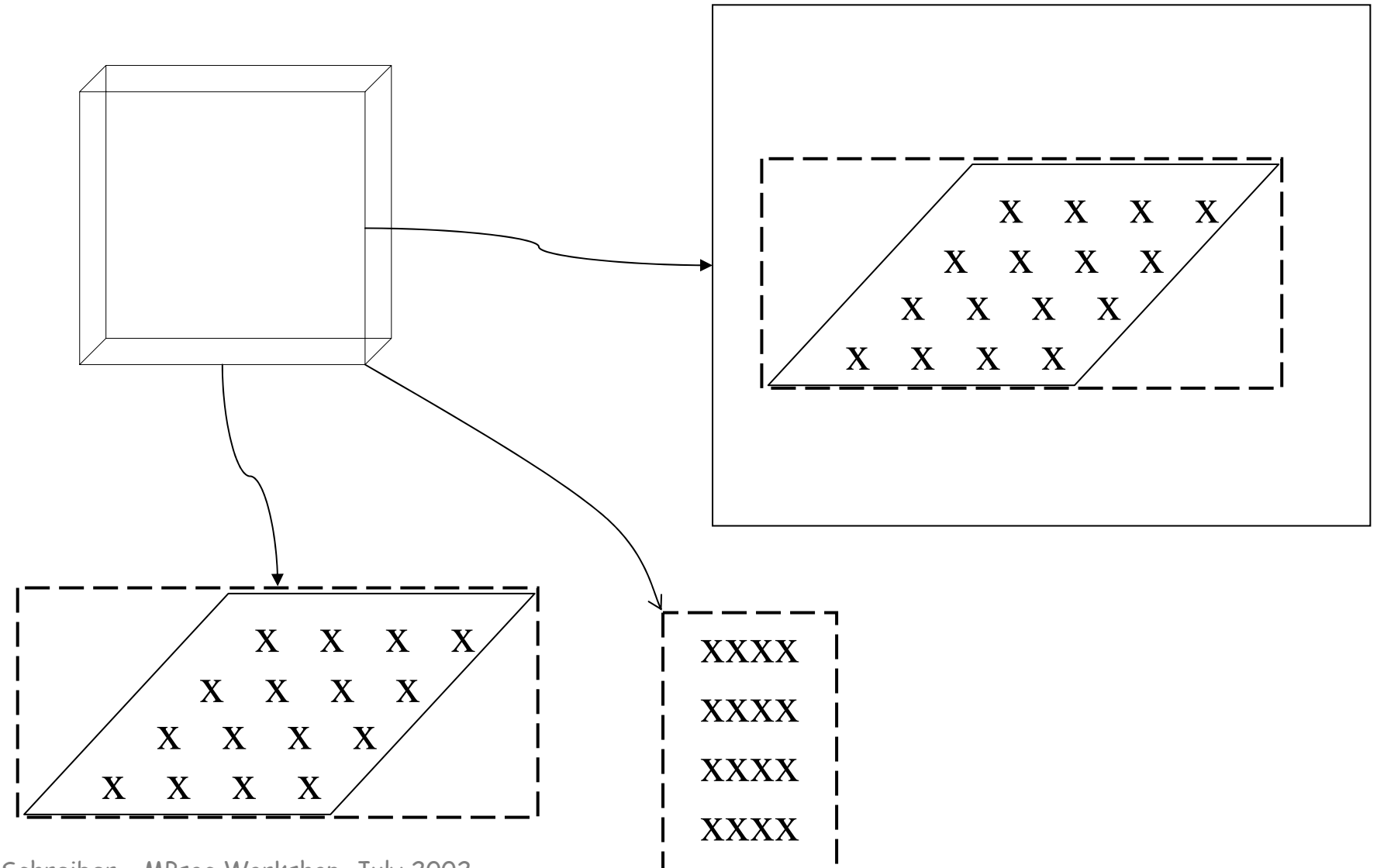
Example: the Affine Image of an Iteration Space



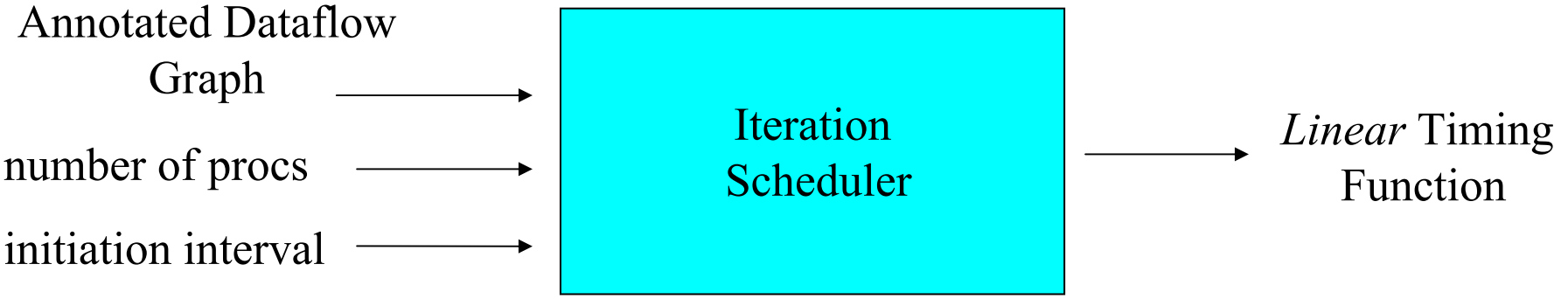
Corrected Estimates

- Published bounds on the size of the image of a Z-polytope are *wrong*
- Our corrections:
  - footprint = iteration space for 1-1 mappings
  - 1-1 if no integer null vector in the iteration space
  - corrected bounds from finding number of iterations that differ by a null vector
  - within 20 percent in practice

Reindexing to Reduce Local Memory



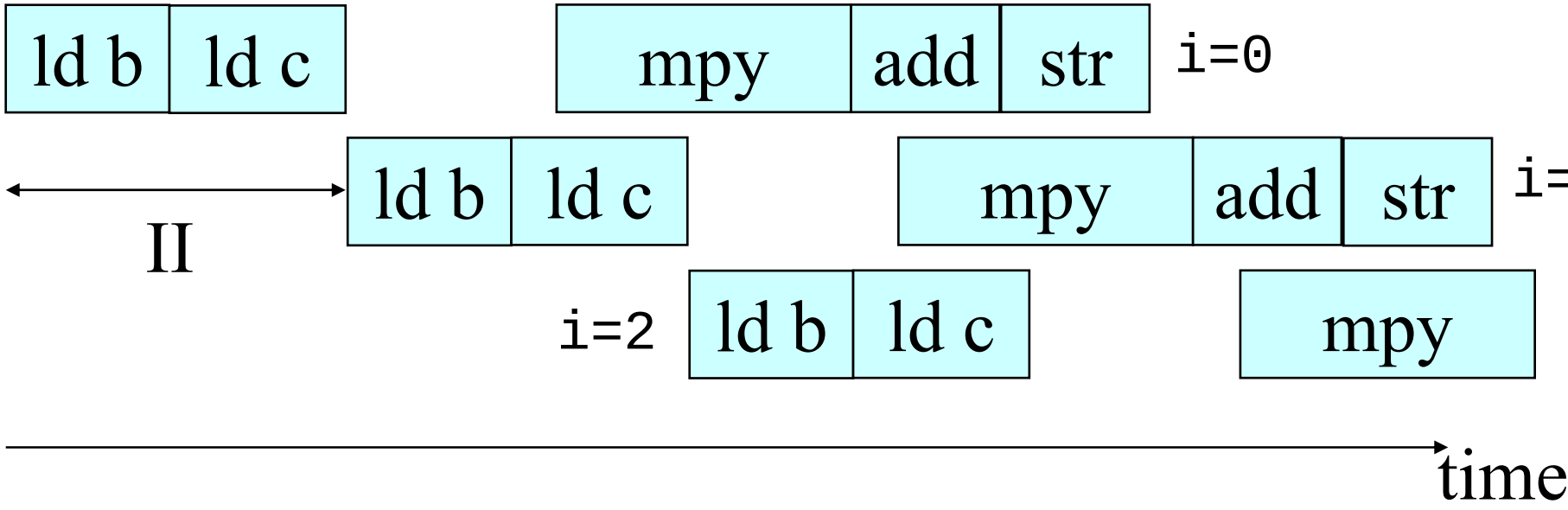
Finding the Parallel Iteration Schedule



- *Processors* a mesh of processors is given
- *Initiation Interval (II)* every processor starts an iteration periodically with period equal to II (*hardware* pipelining)
- *Mapping* clusters of iterations are mapped to each processor
- *Schedule* one iteration per processor every II cycles
- *Honor* data dependence constraints
- *Find the schedule* via efficient direct search method

Hardware/Software Pipelining

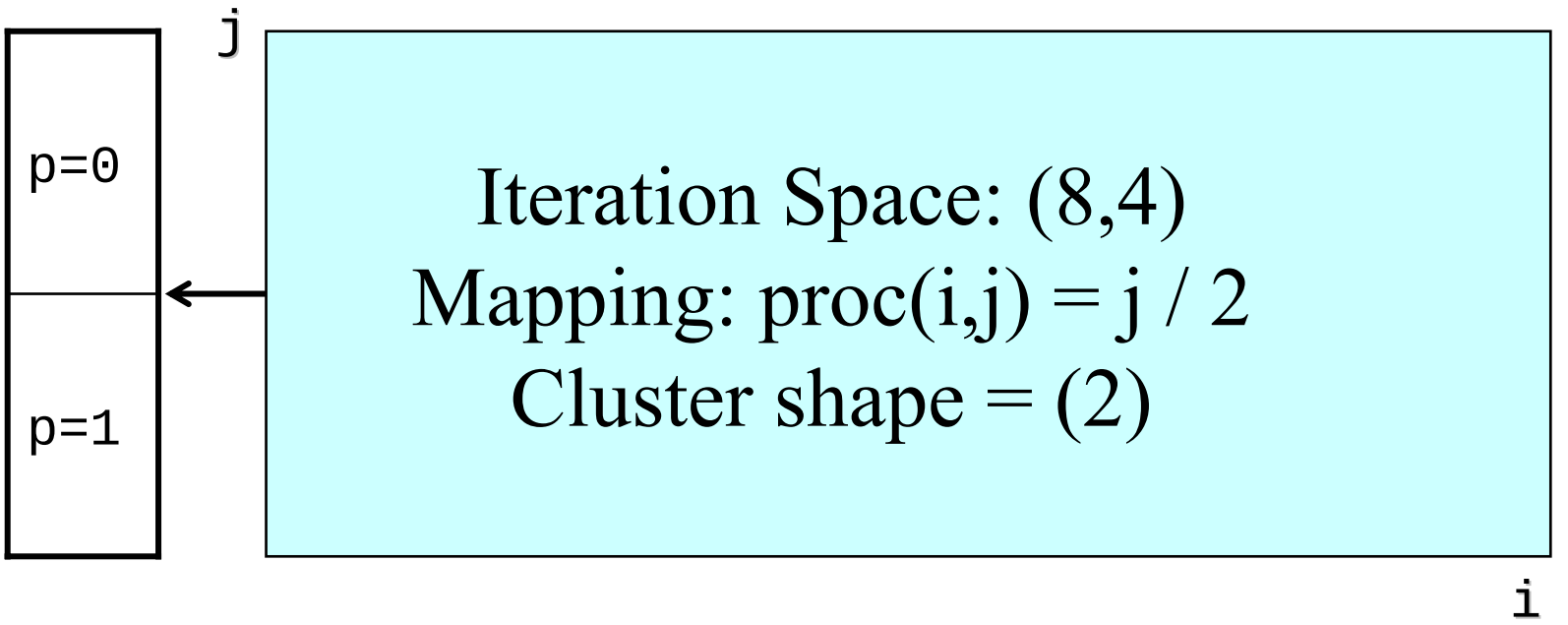
```
for (i=0; i < 100; i++) a[i] += b[i]*c[i]
```



Lower Bounds on II (RecMII, ResMII)

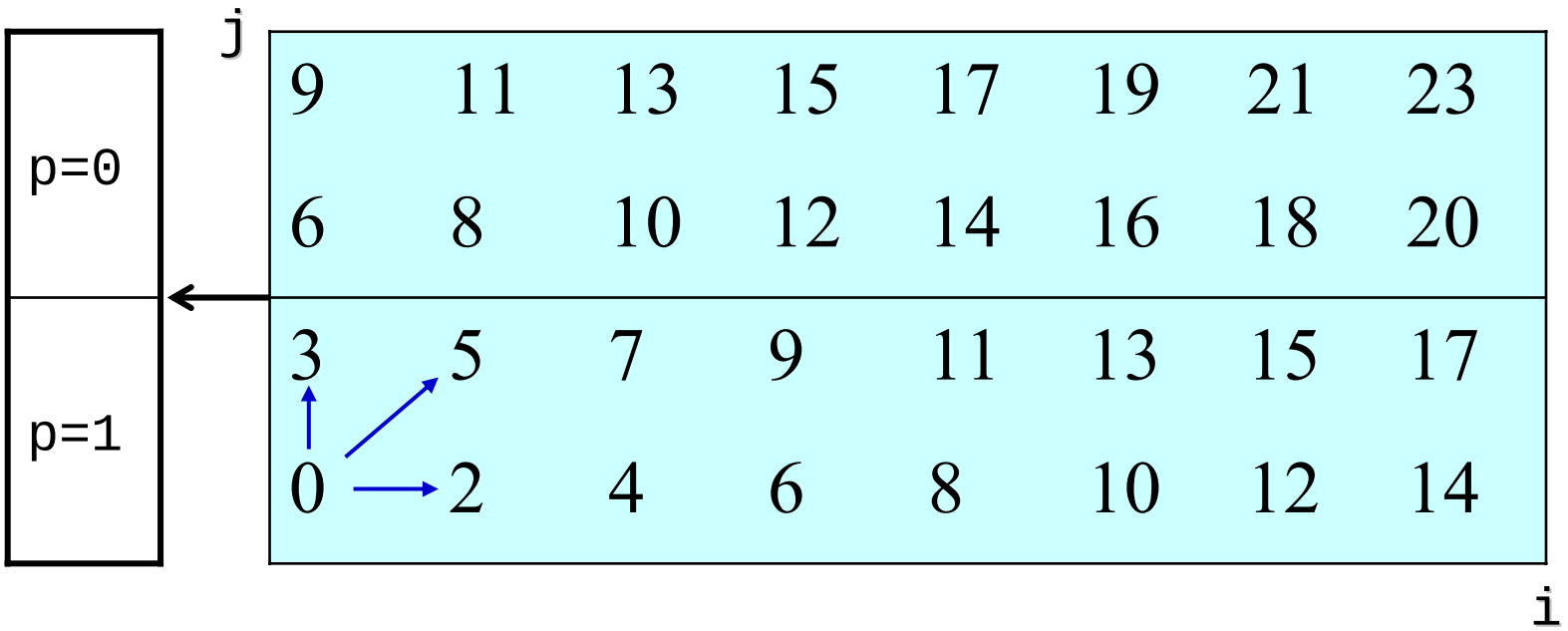
The Mapping of Iterations to Processors

```
for (i = 0; i < 8; i++)
  for (j = 0; j < 4; j++)
   {
     y[i] += w[j] * x[i-j];
   }
```



A Tight Schedule: (i,j) --> 2i+3j

```
for (i = 0; i < 8; i++)
  for (j = 0; j < 4; j++)
   {
     y[i] += w[j] * x[i-j];
   }
```

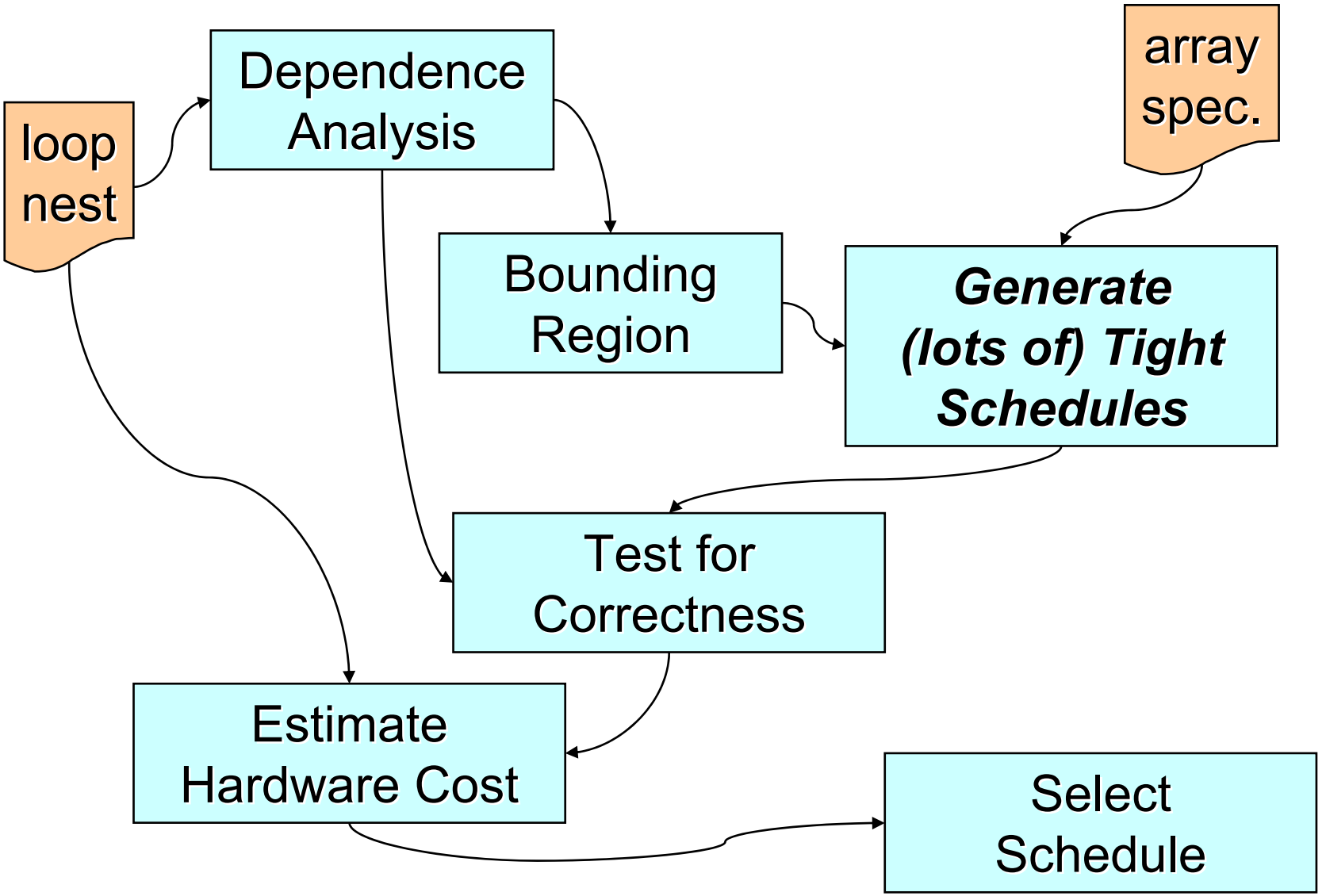


Tight Schedules – Prior Work

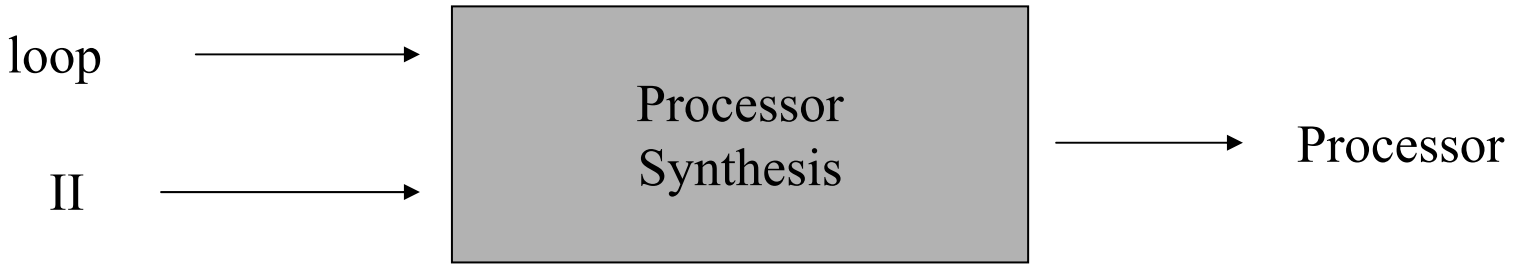
Darte/Delosme, Chen/Megson.

- *GIVEN*: Iteration space, projection direction, linear schedule
- *DETERMINE*: The allowed cluster shapes
- *Tail Wags Dog!*

Constructing the Schedule



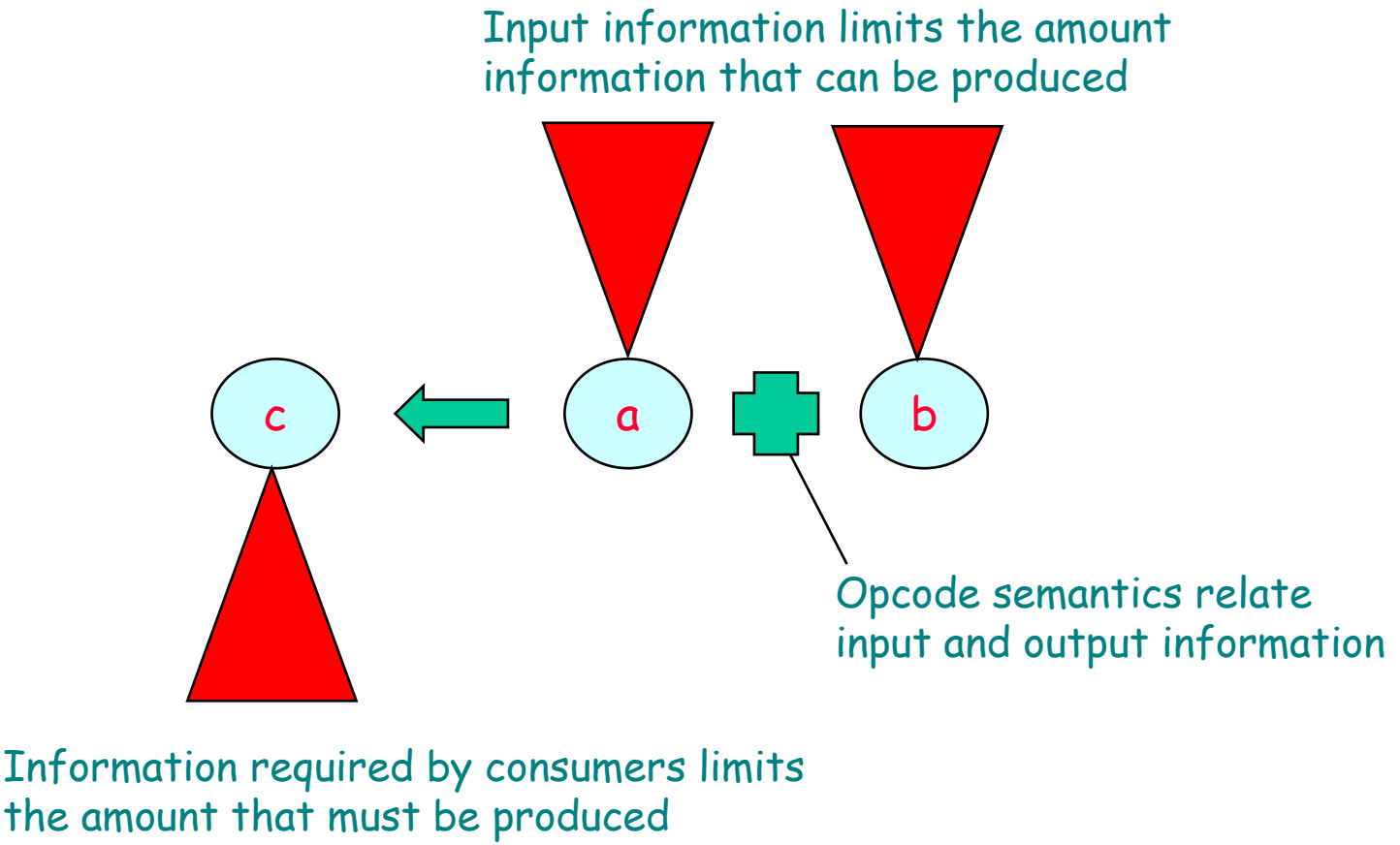
Processor Synthesis



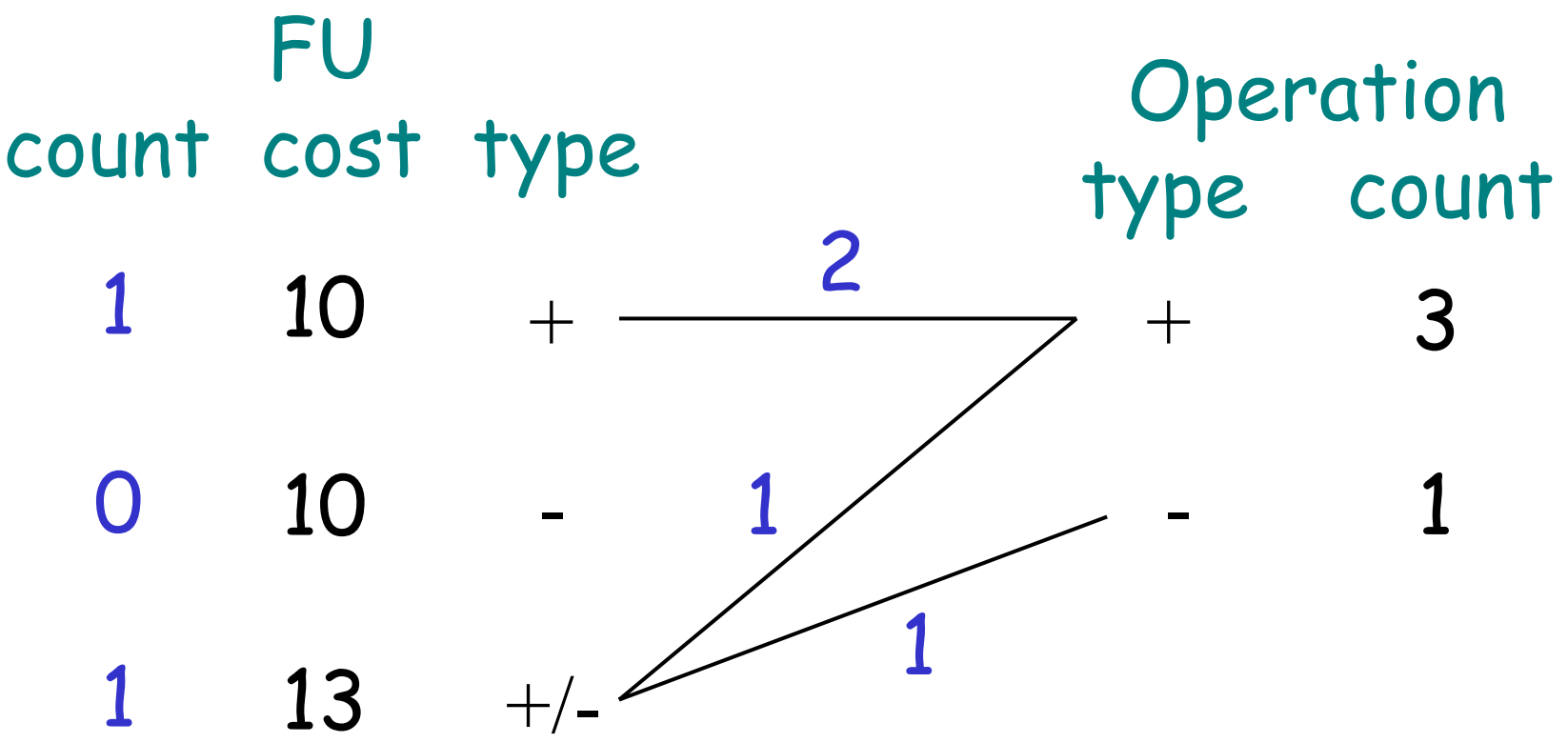
- *Optimize* the loop body
- *Analyze* bitwidth of all values
- *Allocate* the function units
- *Map* operations to function units
- *Schedule* operations
- *Allocate* registers and memory
- *Interconnect* communicating elements

Parallel, custom, designed to spec: EFFICIENT!

Bitwidth analysis - basic idea



Optimal FU allocation

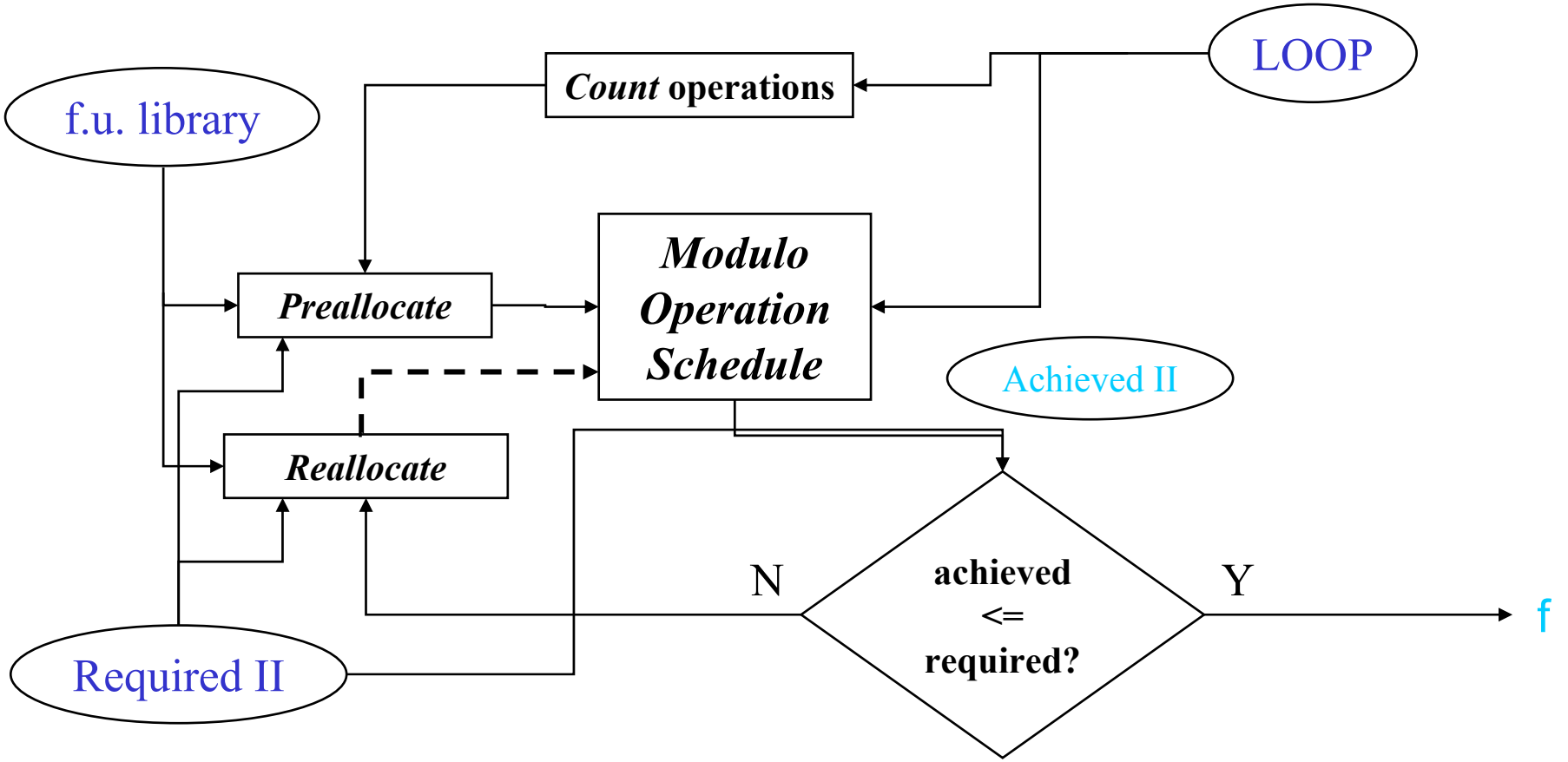


MILP: minimize cost subject to sufficient capacity

Allocation and Op Scheduling

Given: Inner loop and II

Find: Cheapest processor that achieves II on the loop



Conclusions

- Accurate static analysis of memory bandwidth – optimal tiling
- Linear iteration scheduling: solved problem
- Efficient datapath synthesis – a hard problem, good heuristics
- Automatic NPA synthesis is practical
- Automatic synthesis of full embedded systems is feasible, too

Related pubications :

Robert Schreiber, Shail Aditya, Scott Mahlke, Vinod Kathail, B. Ramakrishna Rau, Darren Cronquist, and Mukund Sivaraman.
PICO-NPA: High-level synthesis of nonprogrammable hardware accelerators.
In Journal of VLSI Signal Processing 31: 127-142 (2002).

Shail Aditya, B. Ramakrishna Rau, and Vinod Kathail.
Automatic architecture synthesis of VLIW and EPIC processors.
In Proceedings of the 12th International Symposium on System Synthesis, San Jose, California, pp. 107--113, November 1999.

Alain Darte, Robert Schreiber, B. Ramakrishna Rau, and Frederic Vivien.
Constructing and exploiting linear schedules with prescribed parallelism.
ACM Transactions on Design Automation for Electronic Systems, 7(1), (2002)

Kyle Gallivan, William Jalby, and Dennis Gannon.
On the problem of optimizing data transfers for complex memory systems.
In Proceedings of the 1988 ACM International Conference on Supercomputing, pp. 238--253, 1988.

Scott Mahlke, Rajiv Ravindran, Michael Schlansker, Robert Schreiber, and Timothy Sherwood.
Bitwidth cognizant architecture synthesis of custom hardware accelerators.
IEEE Transactions on Computer-Aided Design of Circuits and Systems, 20(10):1-17, 2001.

William Pugh.
The Omega test: a fast and practical integer programming algorithm for dependence analysis.
Communications of the ACM, 35(8):102--114, 1992.

Patrice Quinton and Yves Robert.
Systolic Algorithms and Architectures.
Prentice Hall International (UK) Ltd., Hemel Hempstead, England, 1991.

B. Ramakrishna Rau.
Iterative modulo scheduling.
International Journal of Parallel Processing, 24:3--64, 1996.

B. Ramakrishna Rau, Vinod Kathail, and Shail Aditya.
Machine-description driven compilers for EPIC and VLIW processors.
Design Automation for Embedded Systems, 4:71--118, 1999.