Accelerating Multi-core Processor Design Space Evaluation Using Automatic Multi-threaded Workload Synthesis

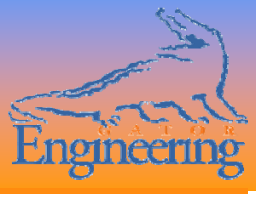
Clay Hughes & Tao Li
Department of Electrical and Computer Engineering
University of Florida

(IISWC 2008)

Presentation By
Clay Hughes

IDEAL Research
(Intelligent Design of Efficient Architectures Laboratory)
Electrical and Computer Engineering
University of Florida

Motivation



- Uni-core systems are becoming increasing rare
- Multi-threaded programming is being widely adopted
- Threaded code can take full advantage of multiple processing elements
- Multiple threads running on multiple processing elements is the prevalent execution paradigm for the next generation of computer systems

CMP Design Space Exploration

- Relies on simulation based performance models
- Inherently more expensive than uni-core processor
  - Shared/private caches, coherency protocols, interconnections, heterogeneity of cores
  - Extra simulator slow down
- Optimizes performance of multi-core oriented applications

Existing Methods

- Machine Learning and Sampling
  - SimPoint [Sherwood et al. 2002]
  - SMARTS [Wunderlich et al. 2003]
- FPGA Acceleration
  - RAMP [Wawrzynek et al. 2007]
- Statistical Simulation
  - HLS [Oskin et al. 2000]
  - [Nussbaum & Smith 2002]
  - [Eeckhout et al. 2003]

Existing Methods - 2

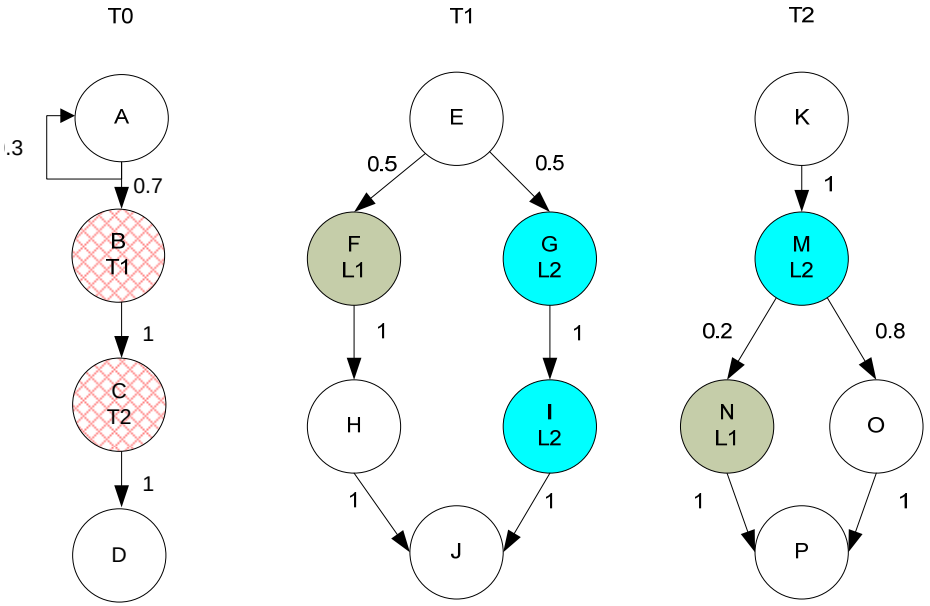
- Synthetic Benchmarks [Bell et al. 2005]
- Capture workload characteristics at low-level
- Build representative instruction trace
- Memory and branch models
- Compilable (portability!)

Contributions

- Synchronized Statistical Flow Graph (SSFG)
- Thread-aware memory model
- Wavelet-based branch model

SSFG

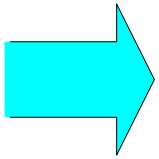
- SFG
  - Nodes represent basic blocks
  - Edges represent transition probabilities
- SSFG
  - Separate SFG for each thread
  - Nodes contain thread-management information
  - Nodes contain synchronization information



Thread-aware Memory Model

```
size_t myUnsigned = 7;

void * myFunction (void *ptr)
{
   unsigned int i = 5;
   unsigned int u_int_1 = 600;
   unsigned int array_1[10] = {0};
...
...
   myUnsigned =  myUnsigned + 1;
   myUnsigned =  myUnsigned * 3;
   myUnsigned =  myUnsigned + 10;
   u_int_1 = u_int_1 - 1;
   array_1[i] = u_int_1;
…
...
}
```

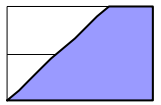


		Starting Address	Stride
SR	mov eax,dword ptr [0x8049770]	8049770	0 0 0 0
	add eax,0x1		
SW	mov dword ptr [0x8049770],eax	8049770	0 0 0 0
SR	mov edx,dword ptr [0x8049770]	8049770	0 0 0 0

.
.
.
.

		Write Stride	Read Stride
W	sub dword ptr [ebp-0x4],0x1	0 4 0 4 0 4	
R	mov edx,dword ptr [ebp-0xc]		0 4 0 4 0 4
R	mov eax,dword ptr [ebp-0x8]		
W	mov dword ptr [ebp+edx*0x4-0x34],eax		

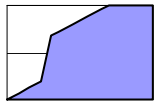
Generate Memory Reference From Distrubtion



("movl %1, %%eax" :"=a"(r_outa) :"m"(shared_memInt[12]); SR

.
.
.
.

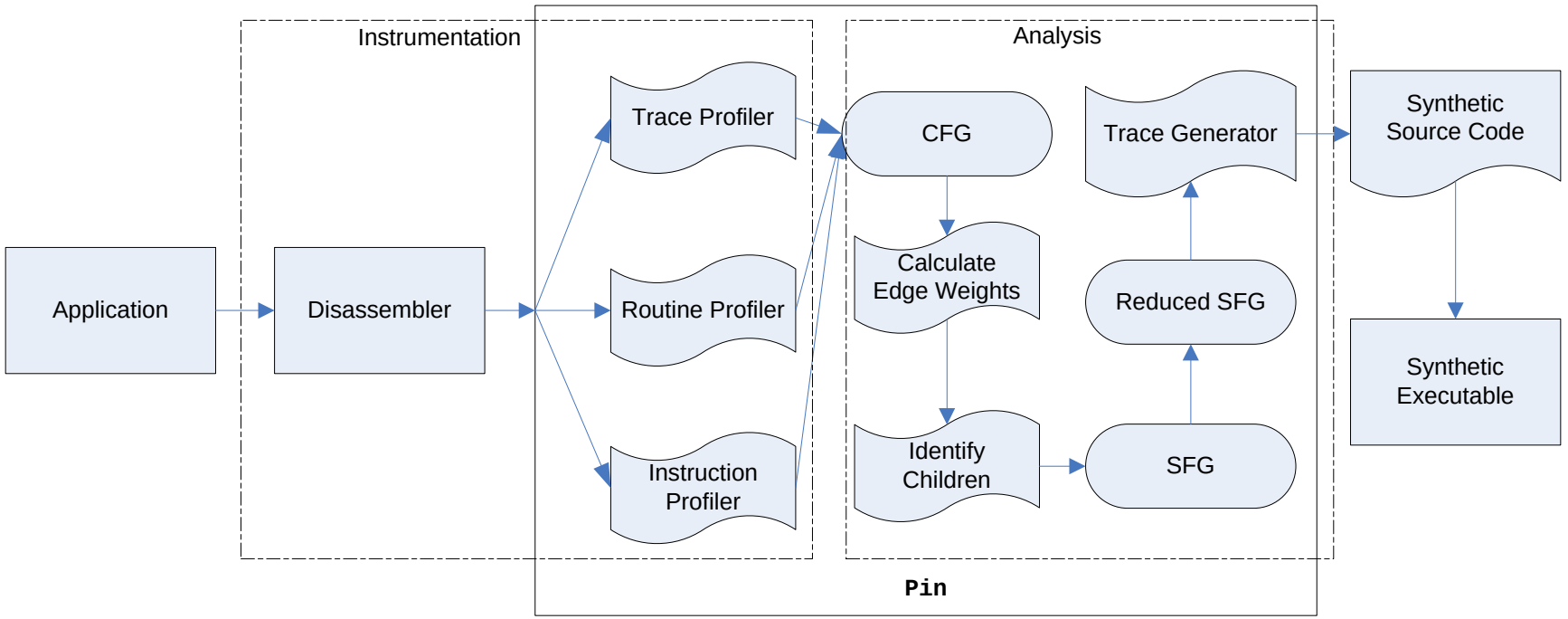
("movl %%eax, %0" :"=m"(memInt[53]) :"a"(r_outa)); W



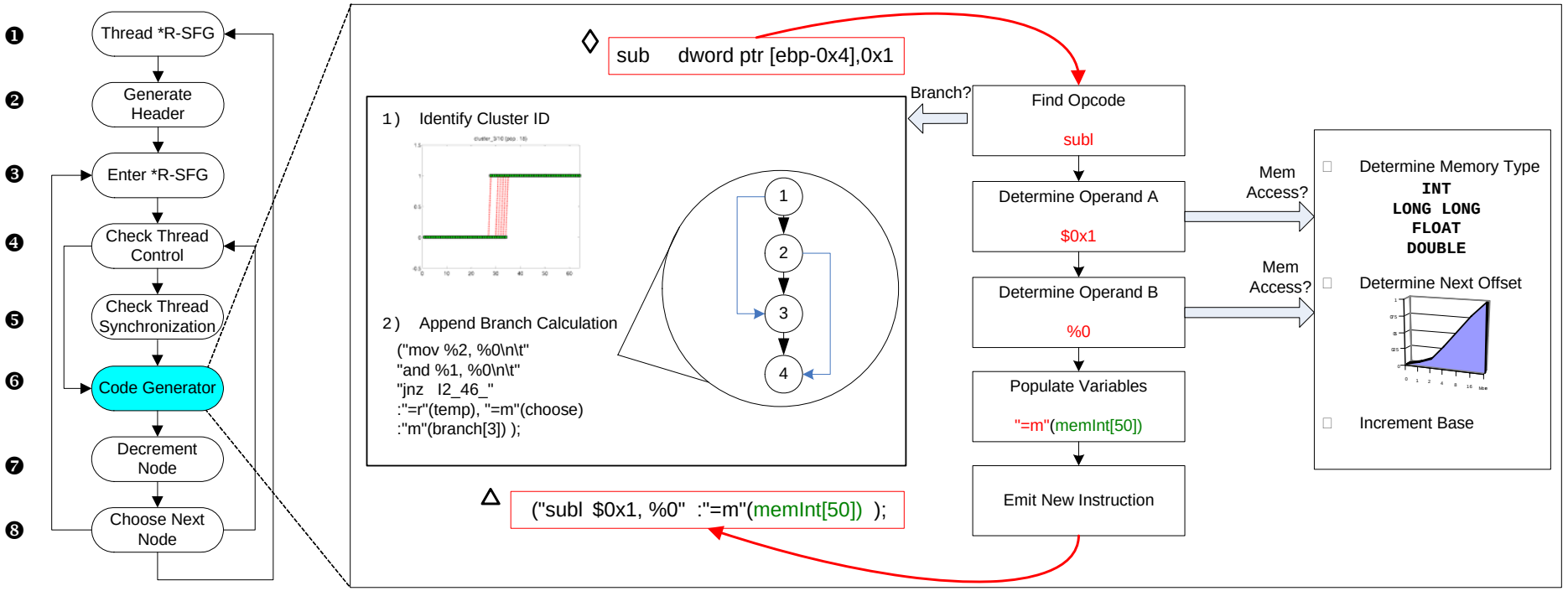
Wavelet-based Branch Model

- Branch transitions stored as bit vectors
- Each bit vector as a time series (e.g. 1 stands for taken and 0 represent not-taken)
- Wavelet analysis is used to extract key patterns
  - 16 wavelet coefficients
  - *k*-mean algorithm classifies branching patterns into clusters based on their similarity
- Using a representative pattern for all branches within the same cluster reduces the overhead of storing each block's branch pattern

Framework (High-Level)



Code Generation



Evaluation: Methods

- Workloads
  - Splash-2
- Tools
  - Pin
  - VTune
- Platforms

Parameter	Platform A	Platform B	Platform C
Processor Type	Pentium 4 - HT	Dual Core Pentium D	Core 2 Quad
Memory	1024MB DDR400	4096MB DDR2-533MHz	4096MB DDR2-533MHz
Storage	80GB SATA HDD	160GB SATA HDD	180GB SATA HDD
Operating System	SuSE 10.01	SuSE 10.01	SuSE 10.2

Evaluation: Accuracy

$$Speedup(\frac{Quad}{Dual})_{Original} = \frac{ExecutionTime(Dual)_{Original}}{ExecutionTime(Quad)_{Original}}$$

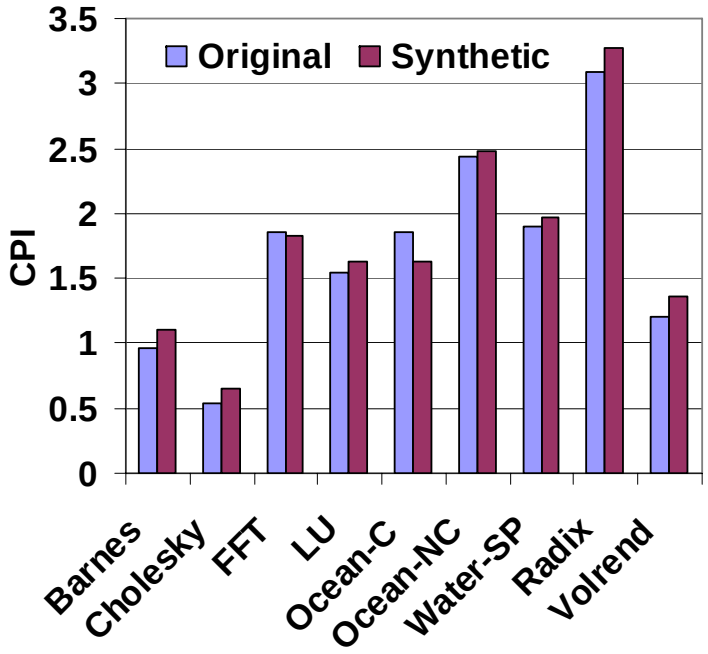
		Barnes	*Cholesky*	*FFT*	*LU*	*Ocean-C*	*Ocean-NC*	*Water-SP*	*Radix*	*Volrend*	**(Cross Bench-mark Error)**
Quad /Dual	Original	2.26	1.75	1.26	1.67	1.23	1.63	1.73	1.84	2.73	
	Synthetic **(Error)**	2.04 **(-9.8%)**	1.92 **(9.7%)**	1.30 **(3.3%)**	1.53 **(-8.6%)**	1.1 **(-10.3%)**	1.53 **(-6.1%)**	1.63 **(-5.6%)**	1.74 **(-5.6%)**	3.05 **(11.7%)**	**(7.9%)**
Quad /HT	Original	2.87	1.8	1.96	3.03	2.8	3.45	2.93	2.28	3.92	
	Synthetic **(Error)**	2.87 **(0%)**	1.98 **(10%)**	2.12 **(8.5%)**	2.64 **(-12.9%)**	2.84 **(1.3%)**	2.95 **(-14.4%)**	2.93 **(0%)**	2.41 **(5.5%)**	4.14 **(5.6%)**	**(6.5%)**
Dual /HT	Original	1.27	1.02	1.55	1.82	2.28	2.12	1.7	1.24	1.44	
	Synthetic **(Error)**	1.41 **(11%)**	1.03 **(0.3%)**	1.63 **(5%)**	1.73 **(-4.7%)**	2.57 **(12.9%)**	1.93 **(-8.8%)**	1.8 **(5.7%)**	1.38 **(11.8%)**	1.36 **(-5.6%)**	**(7.3%)**
	(Cross Platform Error)	**(6.9%)**	**(6.7%)**	**(5.6%)**	**(8.7%)**	**(8.2%)**	**(9.8%)**	**(3.8%)**	**(7.6%)**	**(7.6%)**	

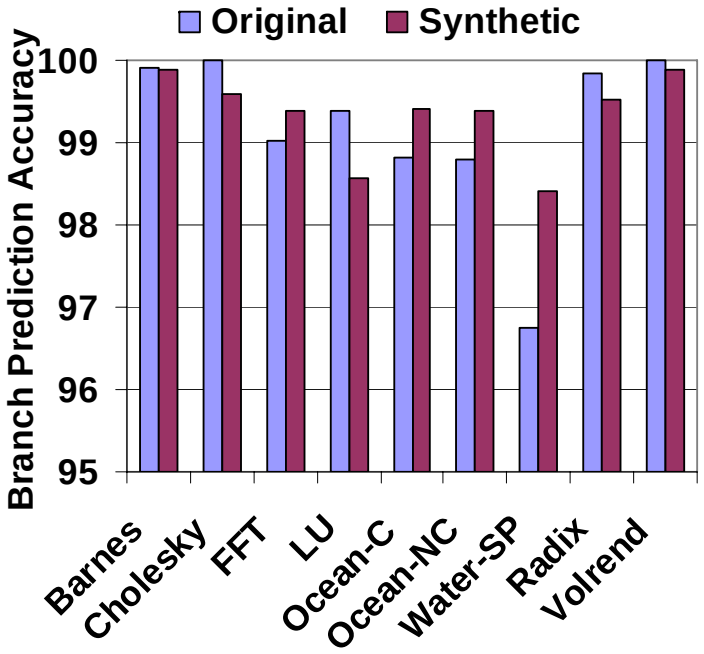
Evaluation: Efficiency

$$\frac{Runtime_{Original}(s)}{Runtime_{Synthetic}(s)}$$

		Barnes	*Cholesky*	*FFT*	*LU*	*Ocean-C*	*Ocean-NC*	*Water-SP*	*Radix*	*Volrend*
HT	2-Thread	30	24	7	10	34	25	413	4	252
	4-Thread	290	145	15	9	21	15	335	12	357
Dual	2-Thread	30	21	8	9	32	26	356	4	286
	4-Thread	261	144	14	9	19	17	316	11	378
Quad	2-Thread	33	19	8	9	31	23	340	4	317
	4-Thread	236	158	14	8	17	16	298	10	422

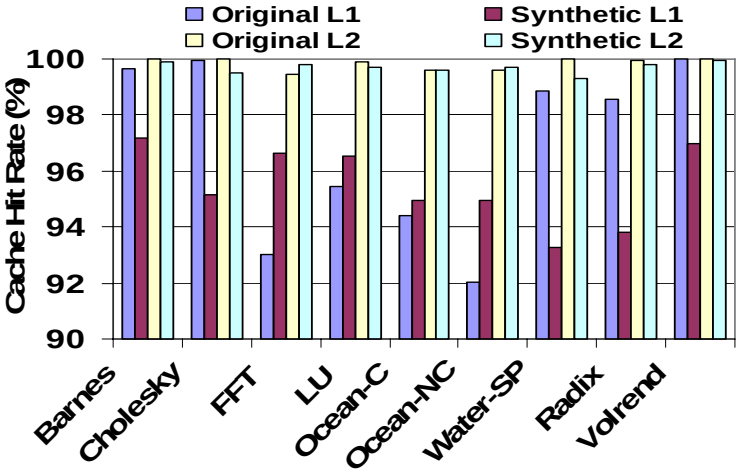
Evaluation: Microarchitecture Events

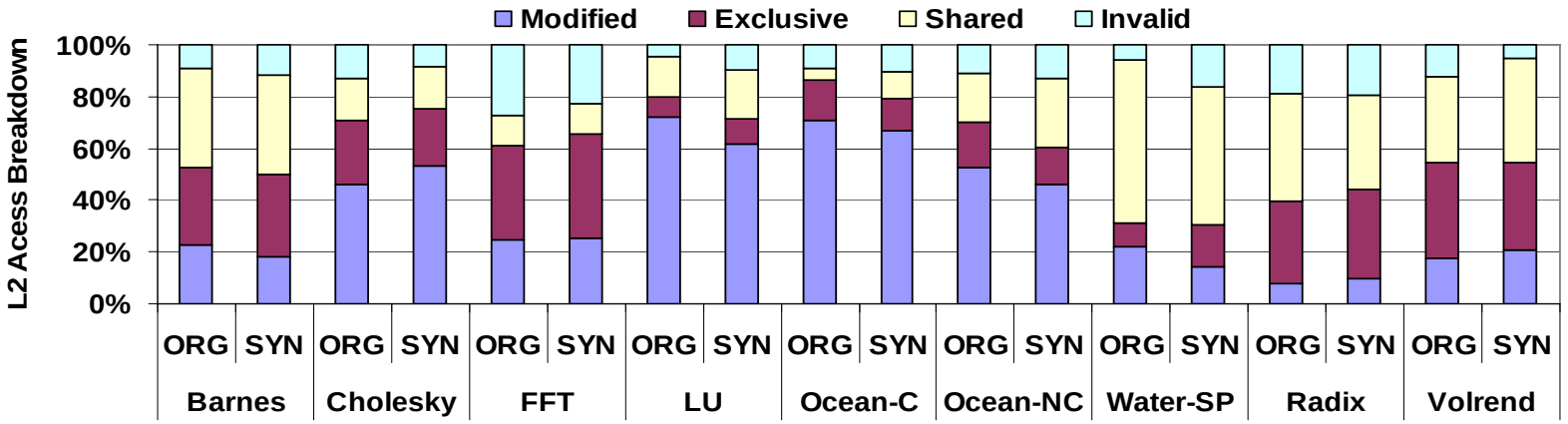




- Maximum CPI error is 12% for 4-thread
- Maximum branch prediction error is 4%

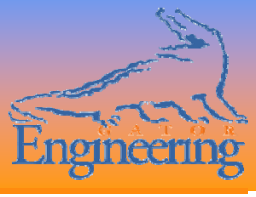
Evaluation: Cache Performance





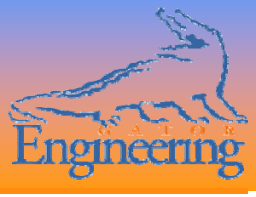
- Maximum cache hit error is 7%

Evaluation: Multi-Core Events



- Locked Operations Impact
  - Measures the penalty caused by locked operations
- Modified Data Sharing Ratio
  - Measures the frequency of modified data sharing due to two threads using and modifying data in one cache line
- Data Snoop Ratio
  - Measures how often a cache is snooped by an adjacent processing element or an external one

Evaluation: Multi-Core Events (2)



		Barnes	*Cholesky*	*FFT*	*LU*	*Ocean-C*	*Ocean-NC*	*Water-SP*	*Radix*	*Volrend*
Locked Operations Impact	Original	0.17%	1.27%	0.82%	0.25%	2.22%	2.62%	0.08%	0.64%	2.3%
	Synthetic Error	**3.5%**	**17.6%**	**-3.2%**	**6.6%**	**-3.2%**	**9.2%**	**-11.4%**	**-2.7%**	**11.7%**
Modified Data Sharing Ratio per 1k Instructions	Original	0.24	0.27	0.17	0.1	0.02	3.1	0.18	0.23	0.23
	Synthetic Error	**-3.5%**	**11.6%**	**7.7%**	**-10%**	**1%**	**-9.2%**	**4.4%**	**2.6%**	**5.6%**
Data Snoop Ratio per 1k Instructions	Original	21	14	46	9	55	75	3	23	3
	Synthetic Error	**-7%**	**-4.8%**	**-7.7%**	**13.2%**	**3.5%**	**6.8%**	**-3.4%**	**-1.6%**	**-5.6%**

IDEAL RESEARCH

Summary

- Multi-core design space is time consuming
  - Joint-optimal designs
- Previous methods suited mostly for single-threaded applications and single PEs
- Synchronized statistical flow graphs allow for multi-threaded synthetic code generation
- Thread-aware memory model
- Wavelet-based branch model

Conclusions

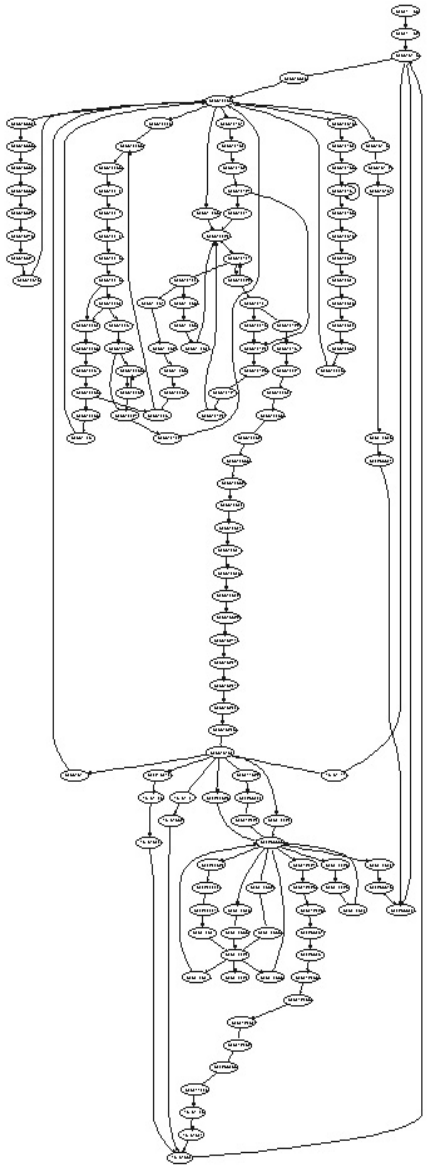
- Preserve high-level program characteristics
- Preserve micro-architecture events
- Preserve complex memory operations
- Reduce runtime by one to two orders

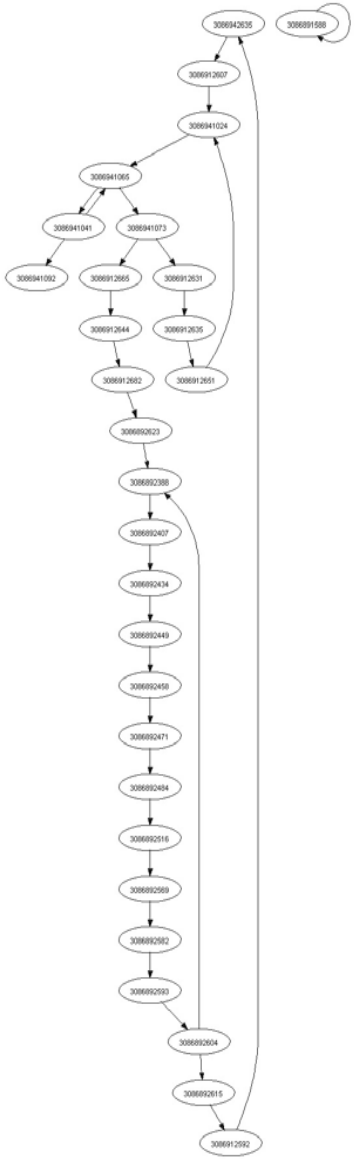
On-going and Future Work

- Support for additional threading/sharing protocols
  - OpenMP
  - HPC
    - UPC
    - MPI
- Support for additional ISAs
  - PowerPC
  - MIPS
- Extend to simulation environment
  - SESC
  - PTLSim
- Commercial and server workload evaluation

Questions

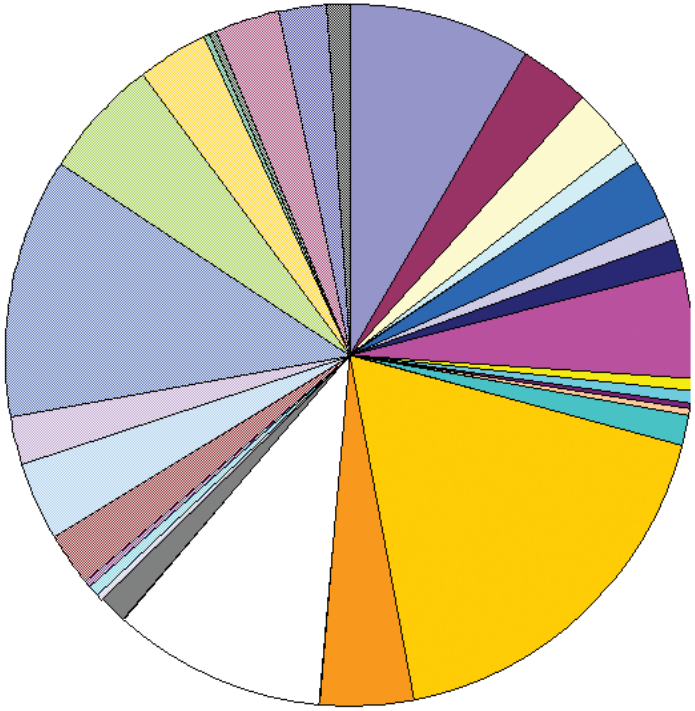
SSFG Example



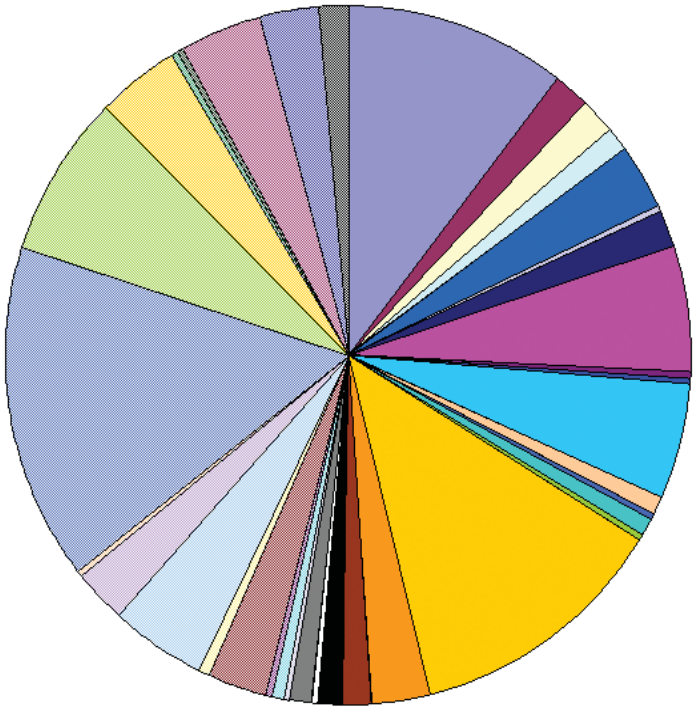


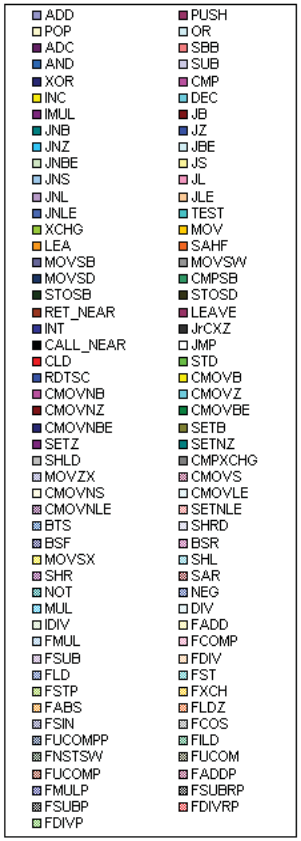
Evaluation: Instruction Mix

Synthetic



Original





- Some instructions merged (calls → branches)