Summary of Summer Research

Kevin Nam

LegUp HLS Group

Cache Simulator

- Simulate program's memory accesses on different caches (different line sizes, levels of associativity)
- Attain miss count, then estimate the total cycles taken by misses
- Use the cache that gives best performance in the synthesized hardware

Results

- Running CHStone benchmarks on Tiger MIPS
- By choosing between 16B, 32B, and 64B line sizes, CHStone sees avg. 5% improvement vs default cache

	adpcm	aes	blowfish	dfadd	dfdiv	dfmul	dfsin	gsm	jpeg	mips	motion	sha
Cycles (16B line size, 8KB cache)	198847	59647	1051432	14601	69997	5002	3158276	45196	4381466	44450	35213	1242875
Cycles (32B line size, 8KB cache)	195107	56414	1054978	13057	68902	4457	3105822	43603	4364256	47440	33647	1234226
Cycles (64B line size, 8KB cache)	192925	54938	1120449	12280	68512	4166	3087084	42920	4780415	50580	32896	1232300
Cycles as % of original	97.02%	92.11%	100.00%	84.10%	97.88%	83.29%	97.75%	94.96%	99.61%	100.00%	93.42%	99.15%

- Default: 8KB 16B line size
- No real effect on fmax (16B = 71.6MHz, 32B = 71.0MHz, 64B = 73.4MHz)
- Increases LEs (16B = 11755 LEs, 32B = 12632 LEs, 64B = 14440 LEs)

Results cont'd

- Cycles further reduced by 2-way set associativity
- Cache simulator finds where increasing cache size will give a large reduction in misses
- Using 2-way SA and a 16KB cache where beneficial, CHStone sees average reduction of 8.2% in cycle count

	adpcm	aes	blowfish	dfadd	dfdiv	dfmul	dfsin	gsm	jpeg	mips	motion	sha
Cycles with 2-way SA	192912	50105	928475	12362	68445	4173	2740180	42961	4213316	42853	32858	1230228
Cycles as % of original	97.02%	84.00%	88.31%	84.67%	97.78%	83.43%	86.76%	95.05%	96.16%	96.41%	93.31%	98.98%

- Avg cache size = 9KB (compared to all 8KB originally)
- Using 16/32/64B line sizes (determined by cache simulator)
- Fmax is reduced due to associativity (~60MHz).

Outcomes of the Cache Simulator Project

- Some performance improvement
- Ability to change the cache size, line size, and associativity in the cache
- Line size has been extended to 128B
- Associativity has been extended to 8 way
- The cache parameters will be studied further in James's memory architecture study

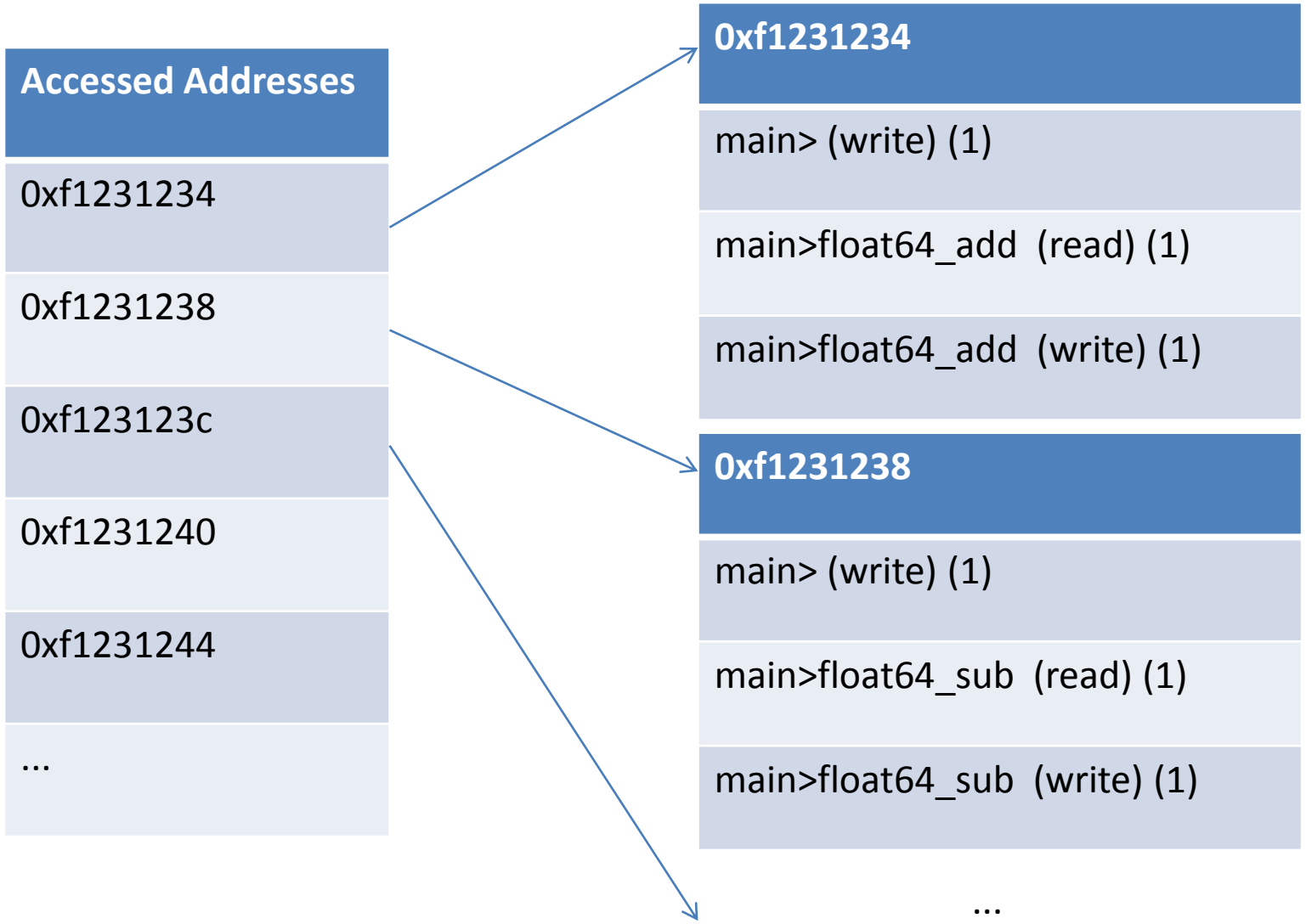
Memory Access Profiler

- **Motivation:** want to determine which functions (accelerators) are parallelizable
- To be parallelizable, a function must be memory independent from the other function it will be running in parallel with
- Idea: profile the memory accesses, determine the memory dependencies between different functions or different invocations of the same function

Memory Access Profiler: How it works

- Use GXEMUL (MIPS emulator) to get an instruction trace of the program
- Go through the instructions, build up a table of all the accessed memory addresses
- For each address, keep track of which functions access it (reads and writes)
- Check all addresses and their accessing functions and check for dependencies

Table of Accessed Addresses



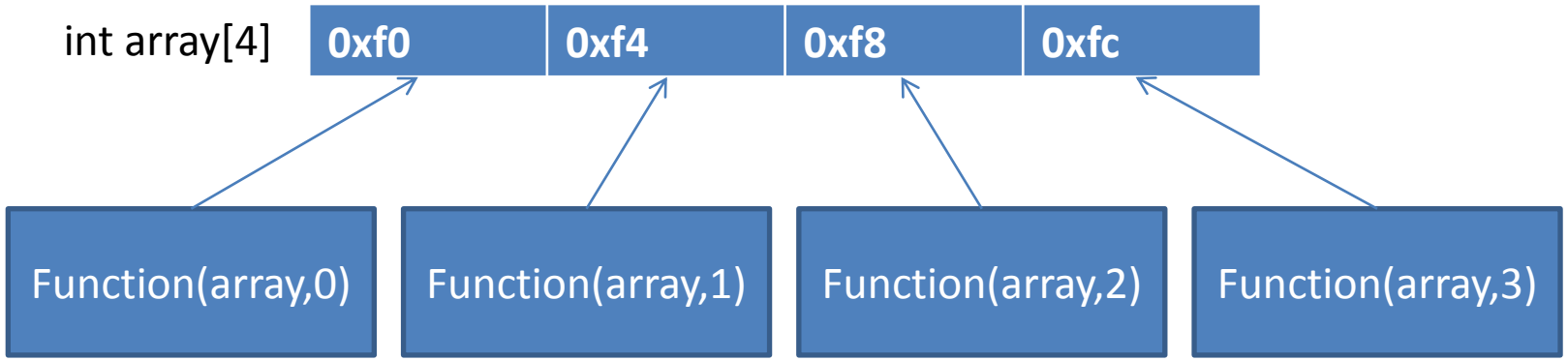
Determining Parallelizable with Self

- Only 1 invocation of the function can access any single address.
- If more than 1 access it, then it is NOT parallelizable with itself

Determining Parallelizable with Self

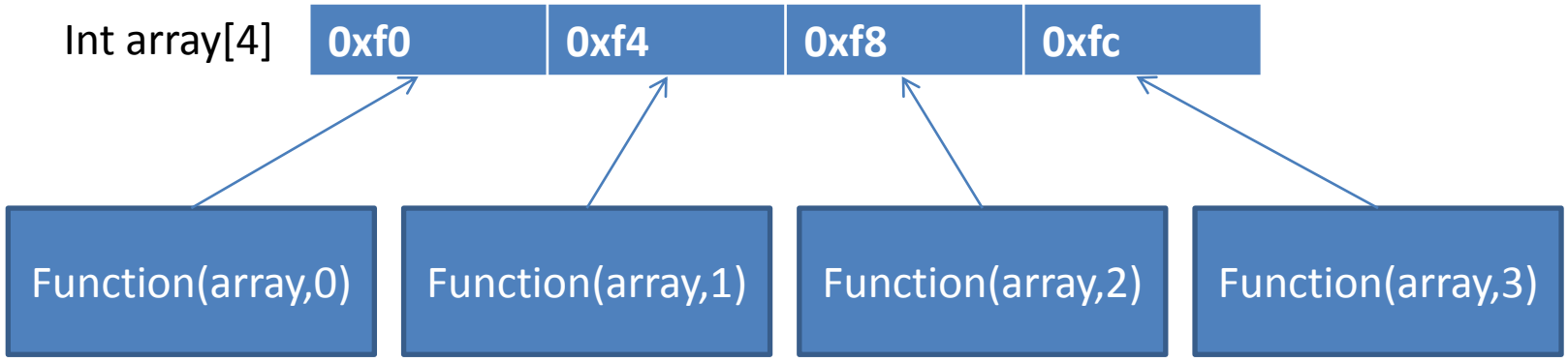
- Example case of parallelizable with self:

```
int main(){
  int array[4];
  Function(array,0);
  Function(array,1);
  Function(array,2);
  Function(array,3);
}
```



Determining Parallelizable with Self

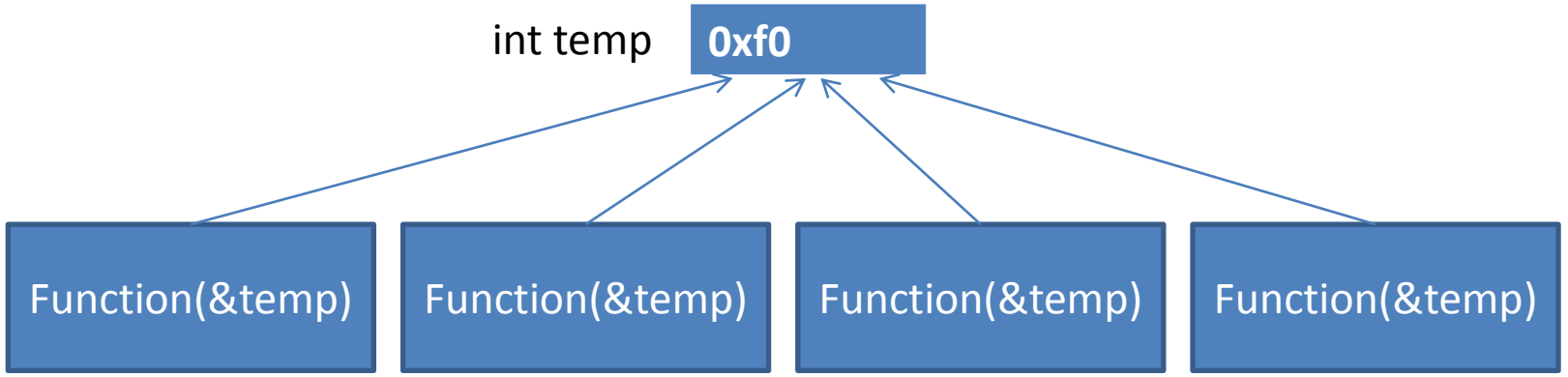
- Example case of parallelizable with self:



0xf0	0xf4	0xf8	0xfc
main>Function (read) (1)	main>Function (read) (1)	main>Function (read) (1)	main>Function (read) (1)
main>Function (write) (1)	main>Function (write) (1)	main>Function (write) (1)	main>Function (write) (1)

Determining Parallelizable with Self

- Example case of NON-parallelizable with self:




```
int main(){
  int temp;
  Function(&temp,2);
  Function(&temp,1);
  Function(&temp,5);
  Function(&temp,2);
}
```

0xf0
main>Function (read) (4)
main>Function (write) (4)

Determining Parallelizable with Other

- Simply check for memory dependencies with other functions

0xf1231234
main> (write) (1)
main>float64_add (read) (1)
main>float64_add (write) (1)

float64_add is parallelizable

Determining Parallelizable with Other

- Simply check for memory dependencies with other functions

0xf1231240
main> (write) (1)
main>float64_add (read) (1)
main>float64_add (write) (1)
main>float64_sub (read) (1)

float64_add is NOT parallelizable with float64_sub

Determining Parallelizable with Other

- Simply check for memory dependencies with other functions

0xf1231244
main> (write) (1)
main>float64_add (read) (1)
main>float64_div (read) (1)
main>float64_sub (read) (1)

float64_add is parallelizable because the other functions' accesses are just reads

Special Cases

- Stack vs Heap variables
- Stack range of each function locality
- Changing memory access pattern for different invocations of the same function
- Passing in arguments to function by value (passing in as a register not address)
- Return value gets assigned to a variable

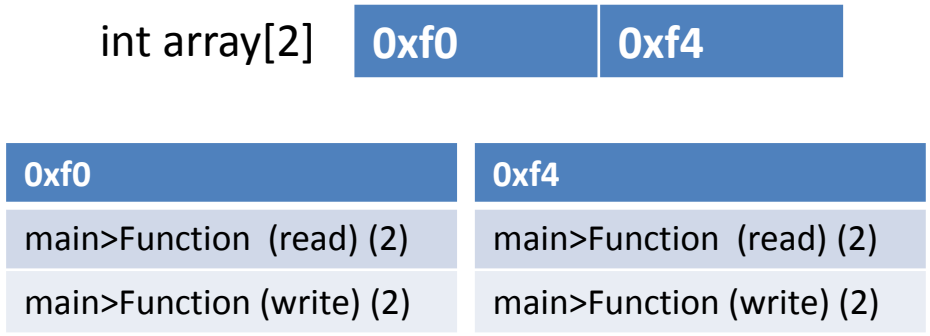
Limitations

- Currently it is a tool to help you find parallelizable functions, not a final go-to solution (user judgement required)
- An example of a limitation:

```
int main(){
  int array[2];
  function(array,0);
  function(array,1);

  ... do other stuff ...

  function(array,0);
  function(array,1);
}
```



Parallelizability of CHStone

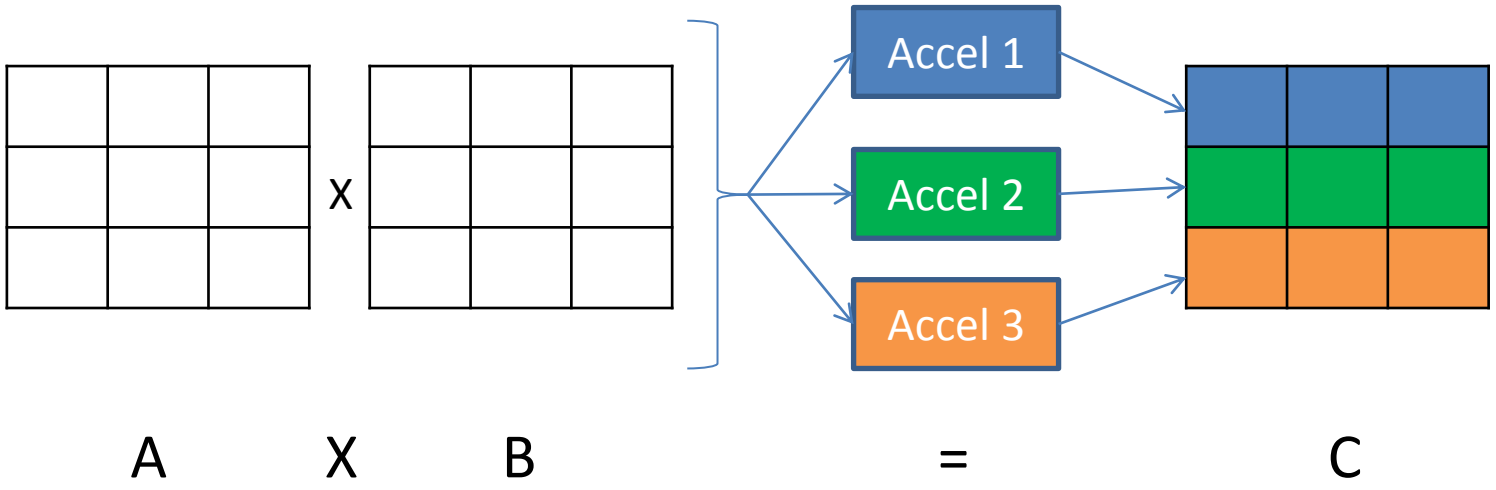
- Access profiler was run on CHStone benches
- Other than the df benches, none worth parallelizing
- A few functions are parallelizable, but they are small functions that don’t take up a large portion of runtime

Parallel Benchmarks

- Needed parallel benchmarks for James's study involving parallel accelerators and memory architecture
- Would like benchmarks with sufficient memory accesses to see the benefit of multi-port cache
- Benchmarks:
  - Matrix Multiply - AES + GSM
  - Box Filter - Adpcm + GSM
  - Bucket Sort - AES + Adpcm
  - Connect6 AI - AES x 3
  - Array addition (James's) - GSM x 3

Matrix Multiply

- When multiplying an nxn matrix by an nxn matrix, the total number of multiplies = n^3
- Split these up into multiple accelerators ex. AxB=C. One accelerator per row



Box Filter

- Uses a box of coefficients to filter an image
- In an nxn image, need to filter nxn pixels
- Split these up into multiple accelerators
  ex. one accelerator per row

Example: 3x3 box

1	1	1
1	2	1
1	1	1

5	4	1	7	8	0	3
3	5	6	4	3	2	3
4	1	5	4	3	4	6
7	7	5	4	4	5	6
8	1	4	4	2	5	4
2	5	4	1	4	5	4
9	4	4	4	2	4	4

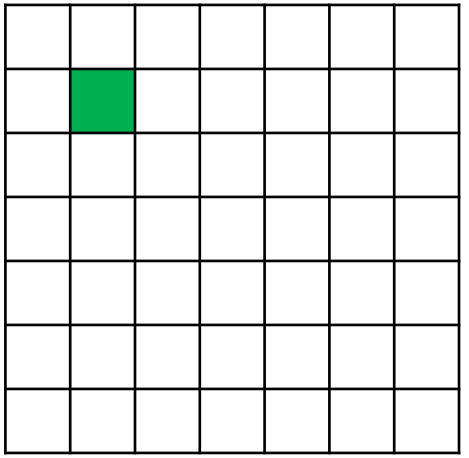
Box Filter

- Uses a box of coefficients to filter an image
- In an nxn image, need to filter nxn pixels
- Split these up into multiple accelerators
  ex. one accelerator per row

Example: 3x3 box

1	1	1
1	2	1
1	1	1

5	4	1	7	8	0	3
3	5	6	4	3	2	3
4	1	5	4	3	4	6
7	7	5	4	4	5	6
8	1	4	4	2	5	4
2	5	4	1	4	5	4
9	4	4	4	2	4	4



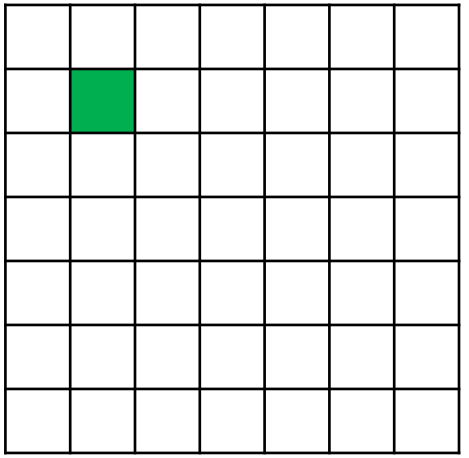
Box Filter

- Uses a box of coefficients to filter an image
- In an nxn image, need to filter nxn pixels
- Split these up into multiple accelerators
  ex. one accelerator per row

Example: 3x3 box

1	1	1
1	2	1
1	1	1

5	4	1	7	8	0	3
3	5	6	4	3	2	3
4	1	5	4	3	4	6
7	7	5	4	4	5	6
8	1	4	4	2	5	4
2	5	4	1	4	5	4
9	4	4	4	2	4	4



Box Filter

- Uses a box of coefficients to filter an image
- In an nxn image, need to filter nxn pixels
- Split these up into multiple accelerators
  ex. one accelerator per row

Example: 3x3 box

1	1	1
1	2	1
1	1	1

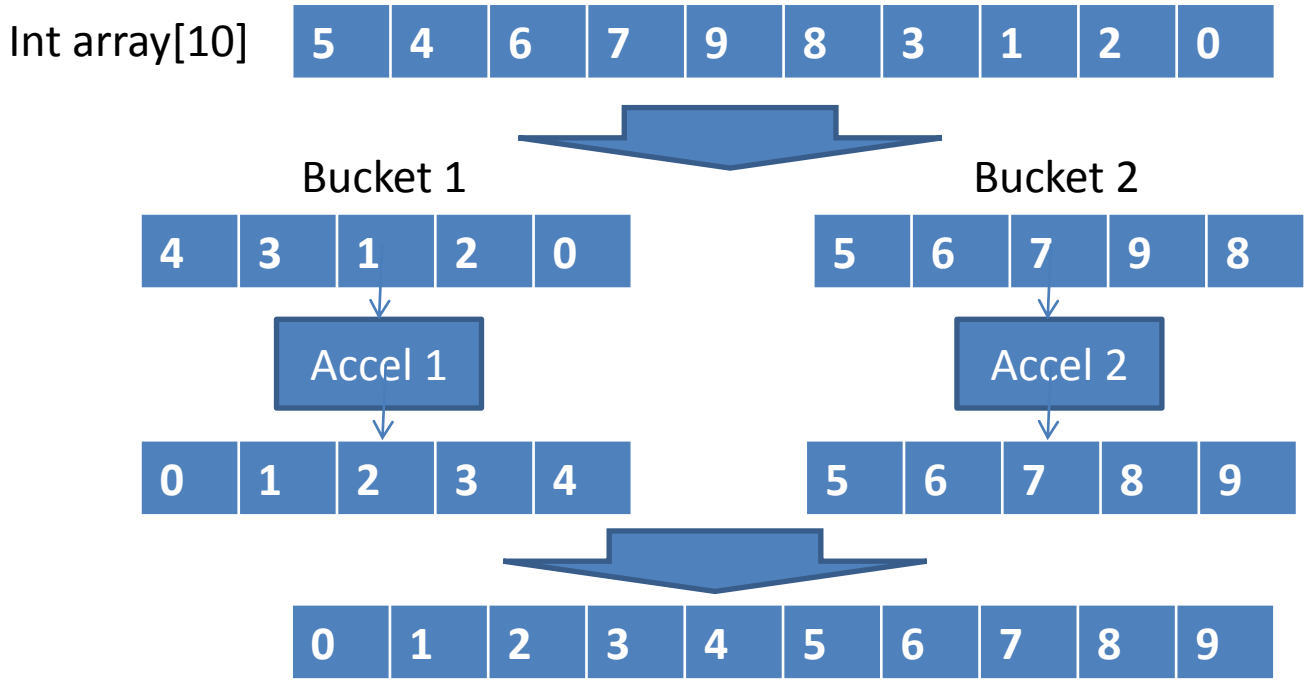
5	4	1	7	8	0	3
3	5	6	4	3	2	3
4	1	5	4	3	4	6
7	7	5	4	4	5	6
8	1	4	4	2	5	4
2	5	4	1	4	5	4
9	4	4	4	2	4	4

	4					

(5*1 + 4*1 + 1*1 + 3*1 + 5*2 + 6*1 + 4*1 + 1*1 + 5*1)/9 = 4.33

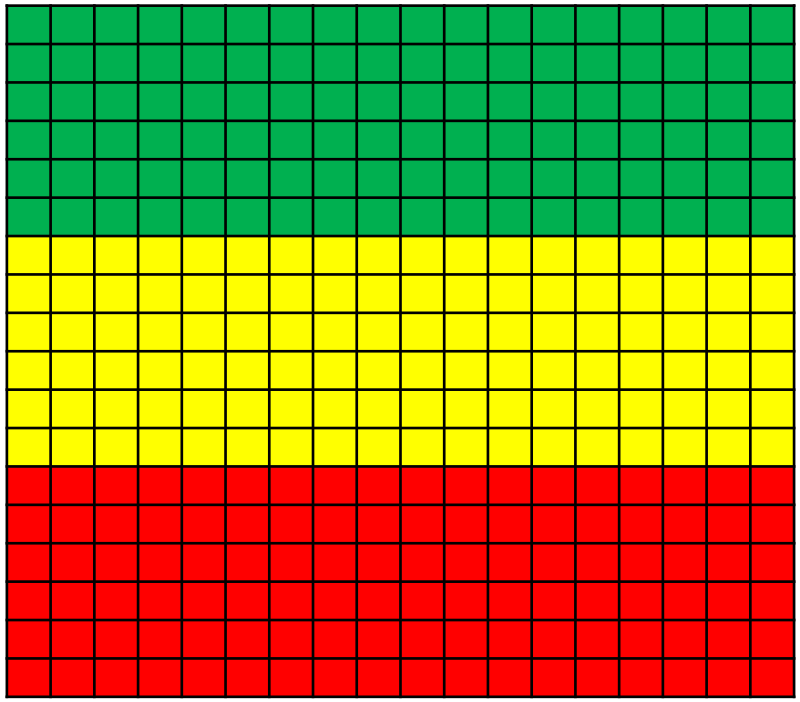
Bucketsort

- Split an n-sized array into m buckets
- Have m accelerators, 1 accel per bucket
- Ex. n = 10, m = 2



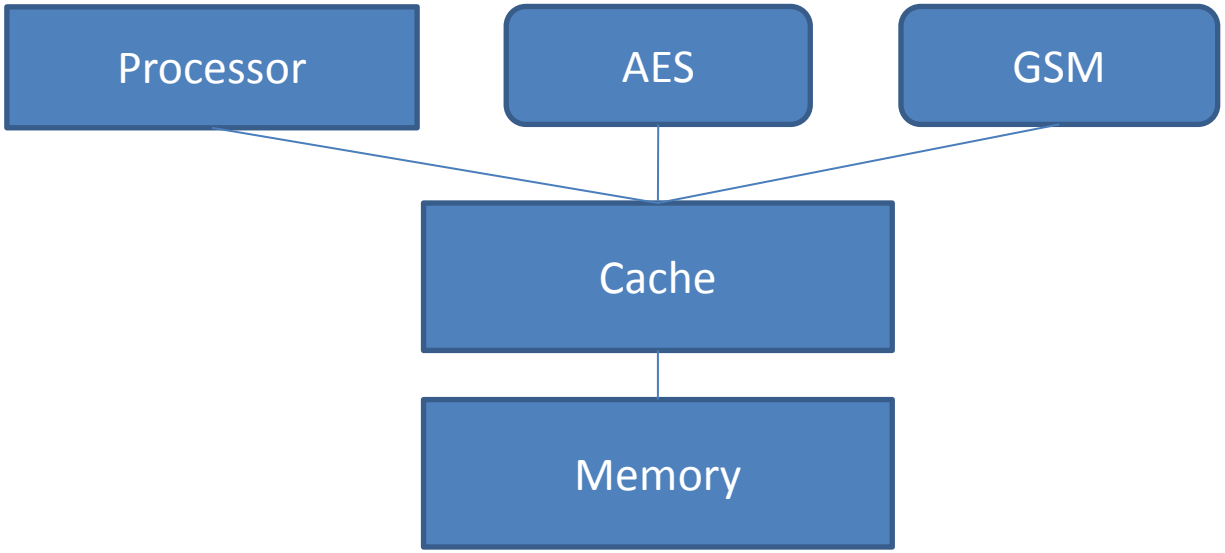
Connect6 AI

- 18x18 board, AI's cost function calculates the cost of each box
- Initialize the board with pieces, check to see if calculated costs are correct, return 0 if correct.
- Split these calculations up into multiple accels
- Ex 3 accels:



Running CHStones in Parallel

- Build an accelerator for each CHStone you want to run in parallel
- Ex. AES + GSM



  - Currently: AES + GSM, and GSMx3.

Current Work

- Currently helping James debug some issues with tiger and parallel accelerators
- Finishing up the CHStone parallel benchmarks