


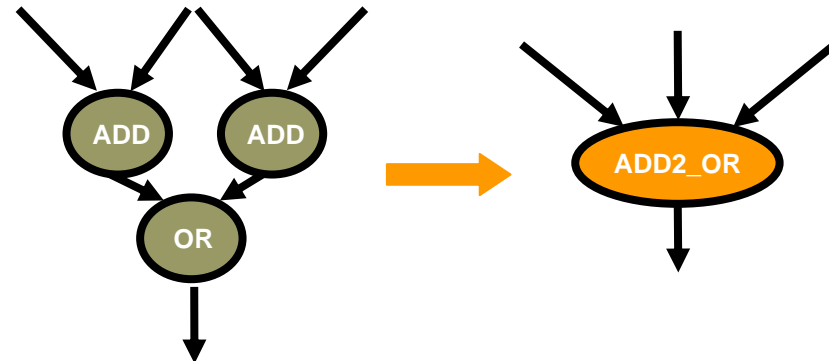
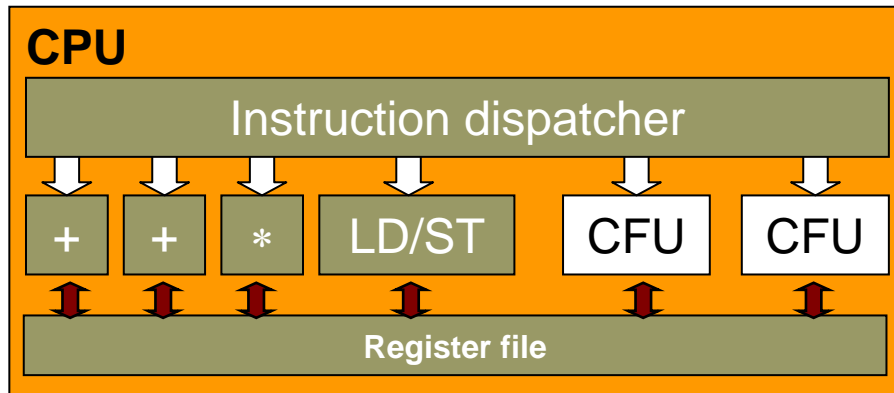
An Efficient Framework for Dynamic Reconfiguration of Instruction-Set Customization



Huynh P. Huynh, Joon E. Sim and Tulika Mitra
School of Computing
National University of Singapore

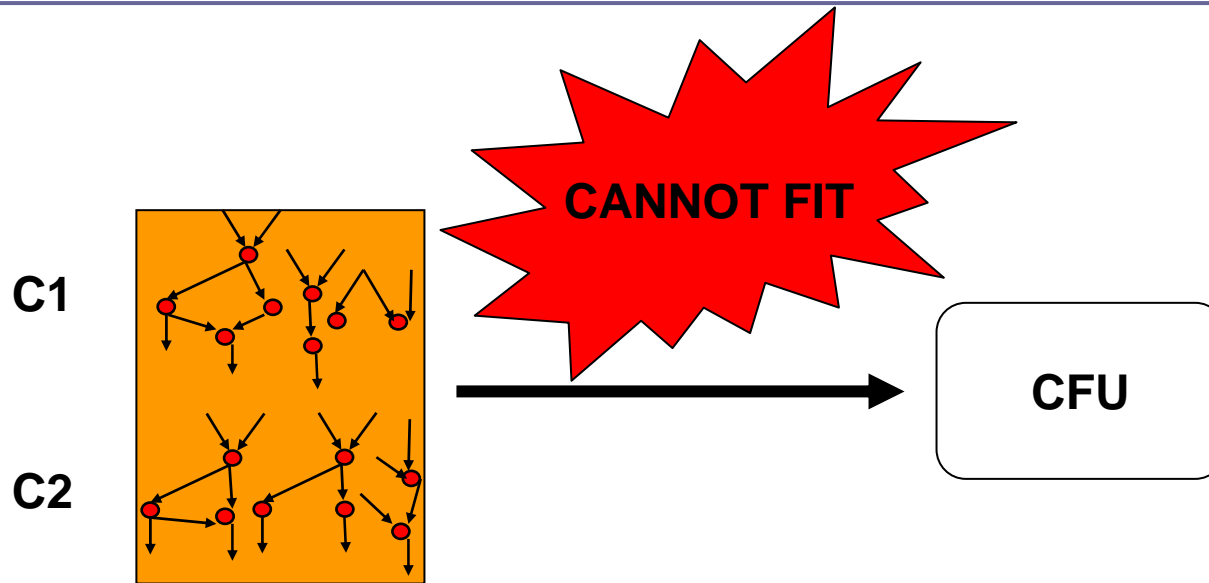
Instruction-set extensible processors

□ Typical architecture



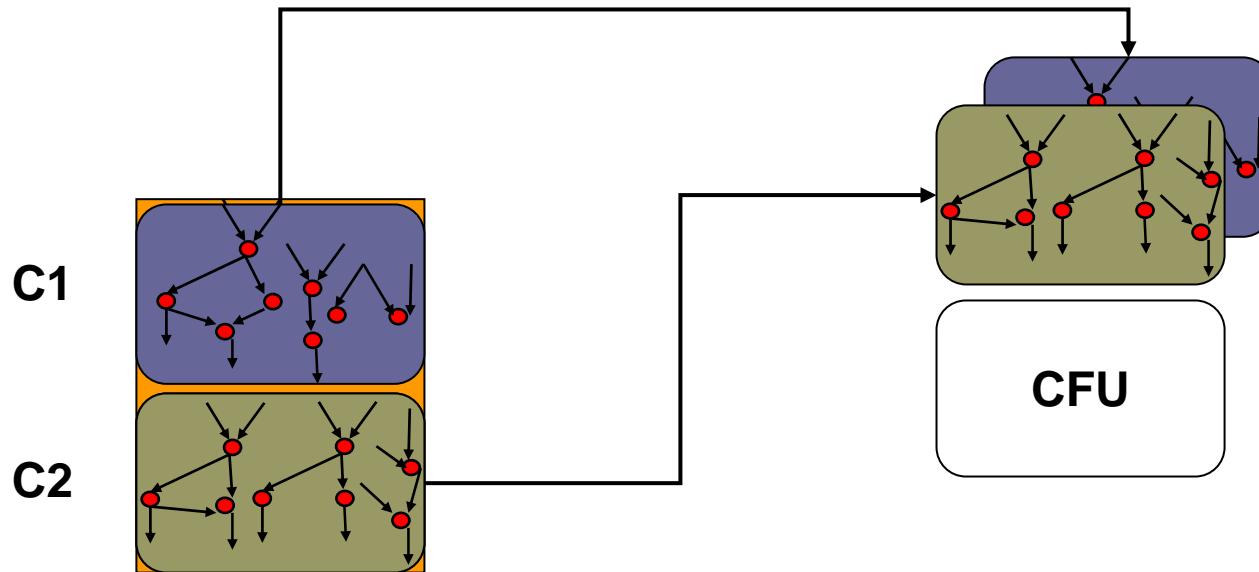
- Examples of commercially available Instruction-set extensible processors: Xtensa (Tensilica), Nios II (Altera)...

Why dynamic reconfiguration?



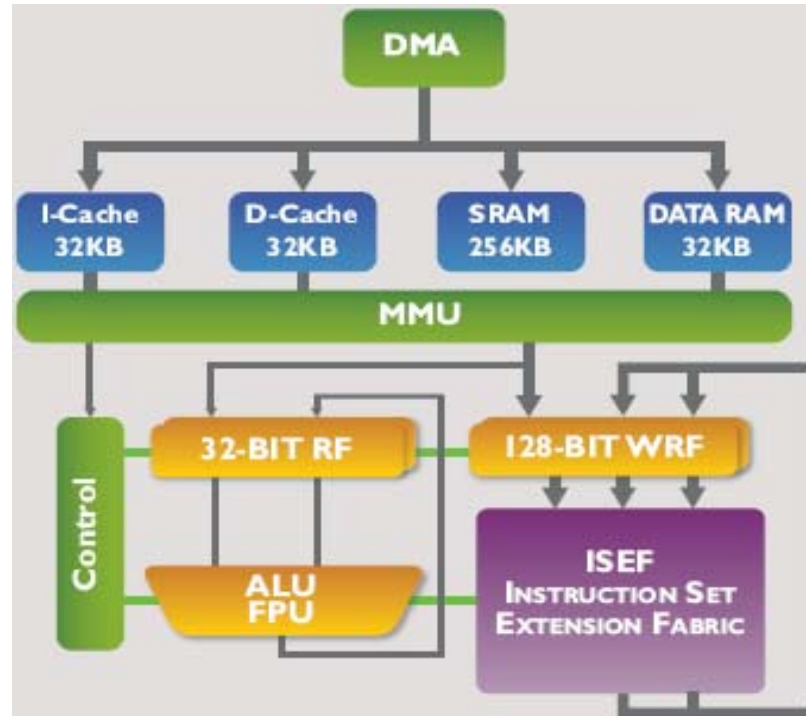
- Select appropriate custom instructions to get best performance gain.
- Partition custom instructions into different configurations so that the reconfiguration cost is minimized.

Why dynamic reconfiguration?



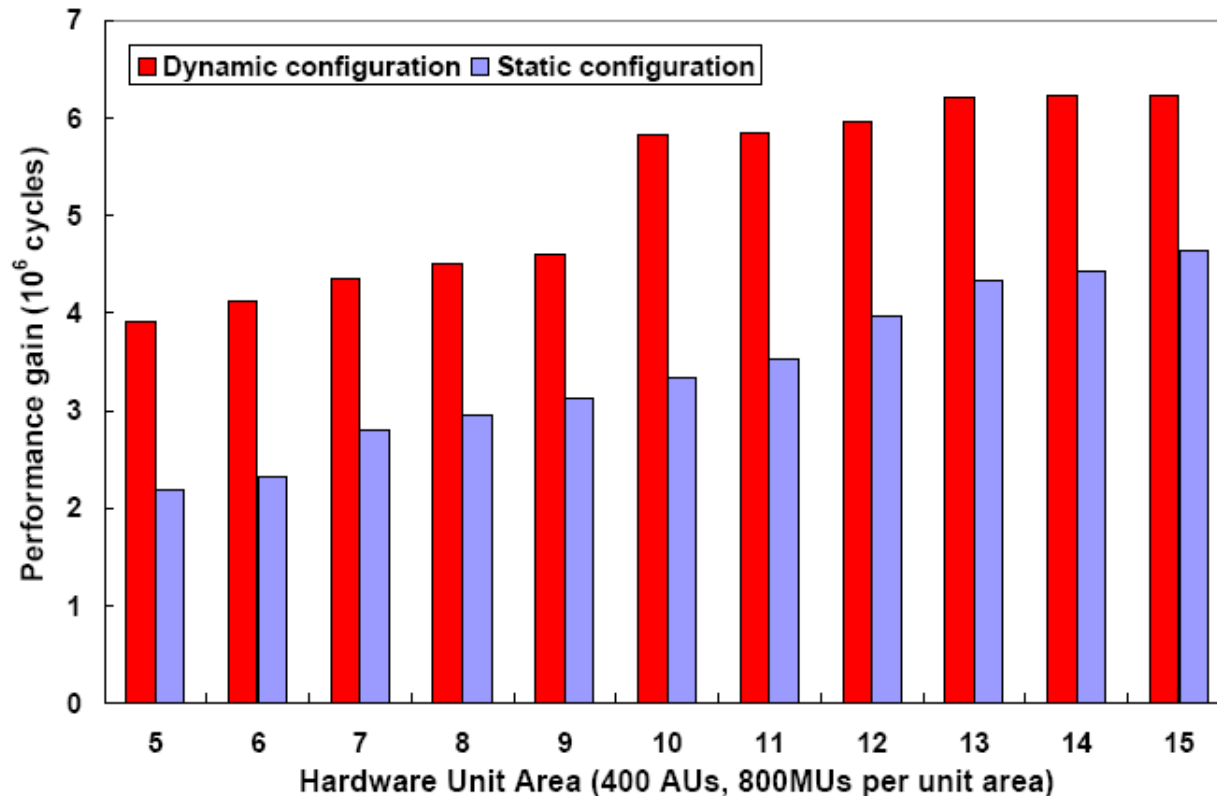
- Select appropriate custom instructions to get best performance gain.
- Partition custom instructions into different configurations so that the reconfiguration cost is minimized.

Stretch 5 – A Dynamically Reconfigurable Custom Instructions Processor



Instruction Set Extension Fabric
is run-time configurable and reloadable.

Dynamic Reconfiguration versus Static Configuration



JPEG implemented in Stretch 5

Custom Instruction-Set for Loop Kernels

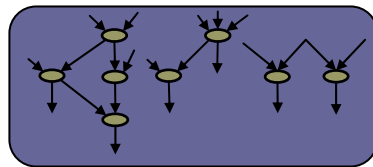
```
int main()
```

```
{
```

```
  for(...)
```

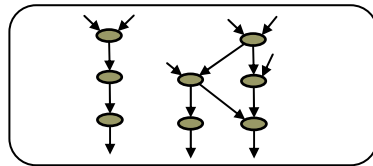
```
  {
```

```
    ...
```



CIS-1

```
    ...
```



CIS-2

```
  }
```

```
    ...
```

```
}
```

```
}
```

```
...
```

```
}
```

```
}
```

Accelerate **compute-intensive loops** with custom instruction-set (CIS).

Dynamic Reconfiguration - Temporal Partitioning

➔ int main()
{

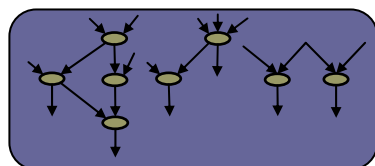
{

➔ for(...)

{

➔

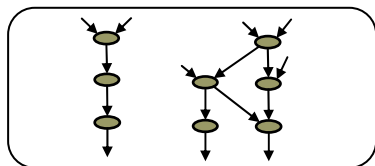
...



CIS-1

➔

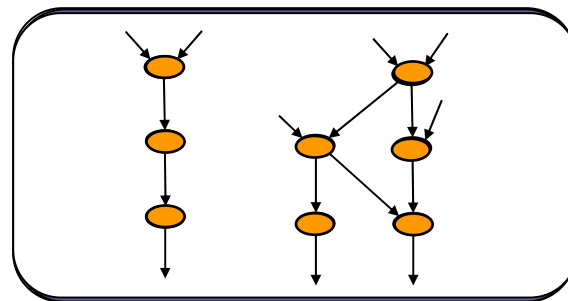
...



CIS-2

}

}



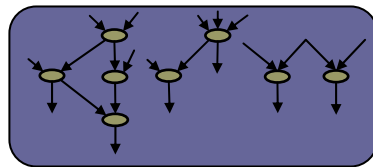
Spatial Partitioning

```
int main()
```

```
{
```

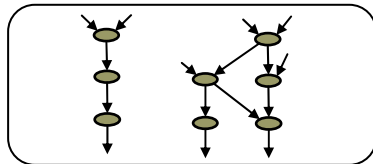
```
  for(...)
```

```
  {
```



CIS-1

```
  ...
```

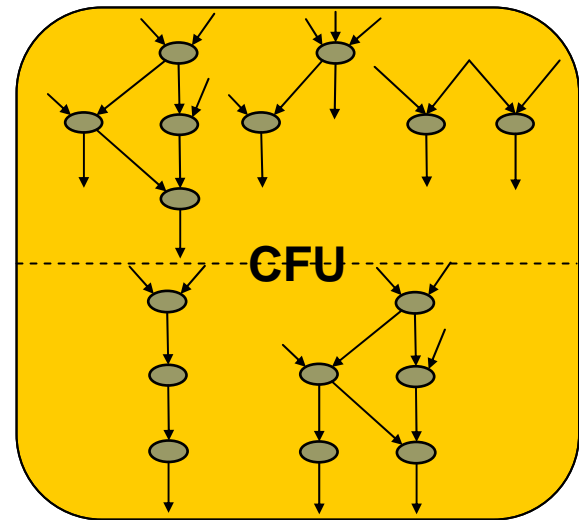


CIS-2

```
  ...
```

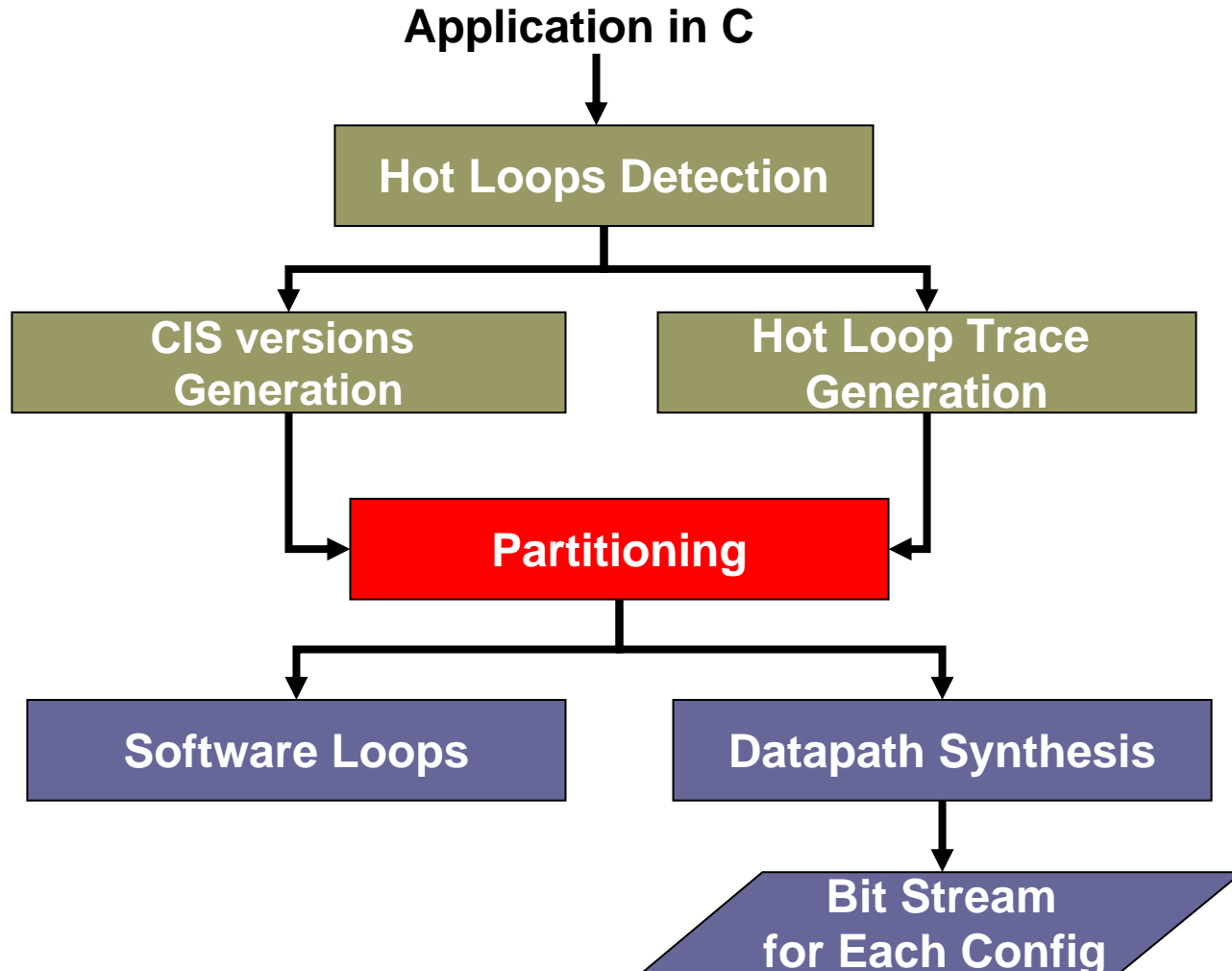
```
  }
```

```
}
```



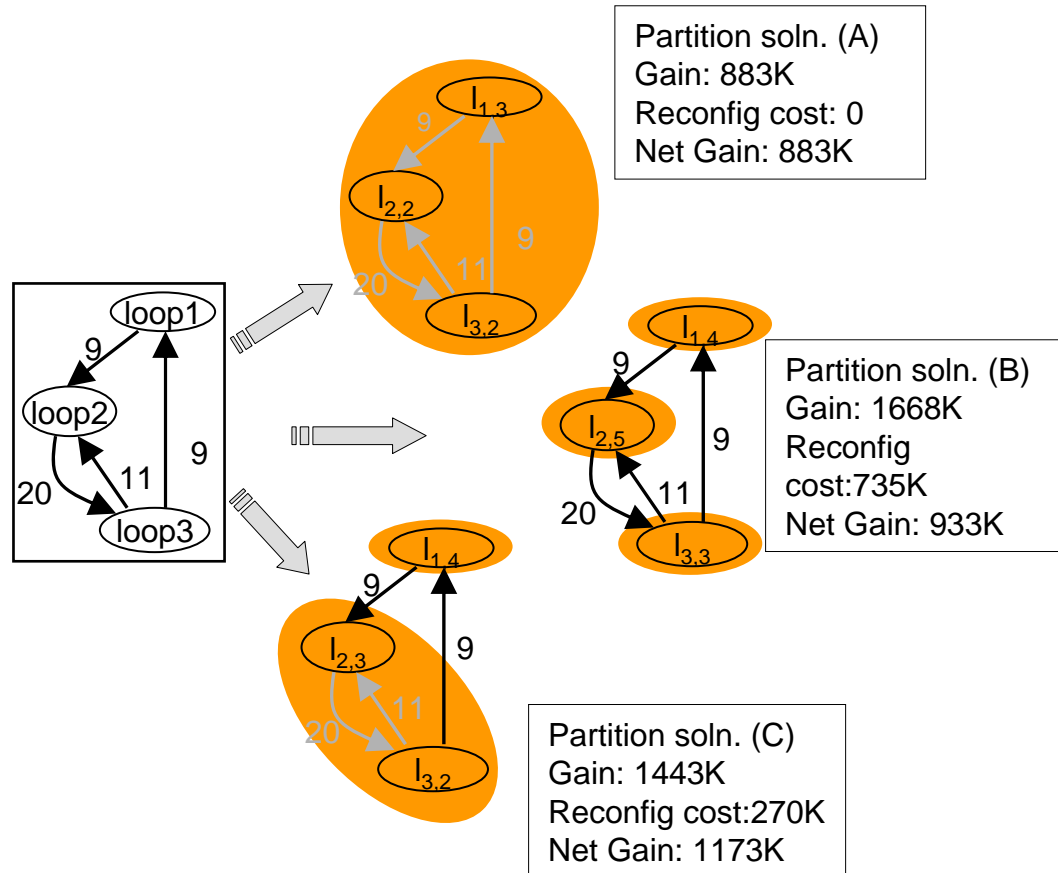
Allows CISs corresponding to **more than one loop** to be placed within **a single configuration**

System Design Flow



Motivating Example

Loop	Version	Area (#AU)	Gain (K cycles)
loop1	1	0	0
	2	257	111
	3	301	160
	4	1612	563
loop2	1	0	0
	2	761	230
	3	1041	387
	4	1321	426
	5	2004	556
loop3	1	0	0
	2	967	493
	3	1249	549



Main Issues in Partitioning Problem

1. Optimal number of configurations k .
2. Temporal partitioning of loop kernels into k configurations.
3. Spatial partitioning of loop kernels in each configuration by selecting CIS version for each loop.

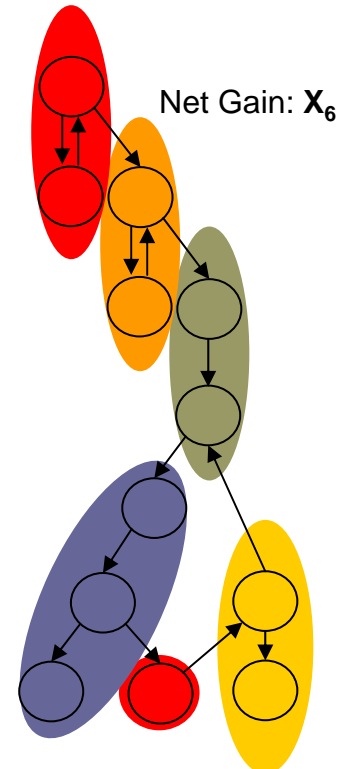
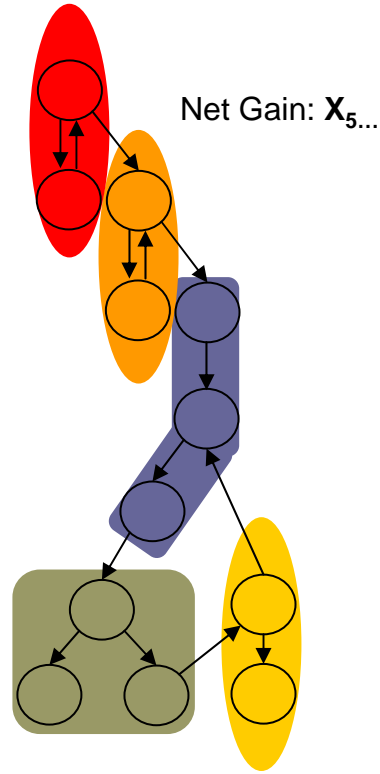
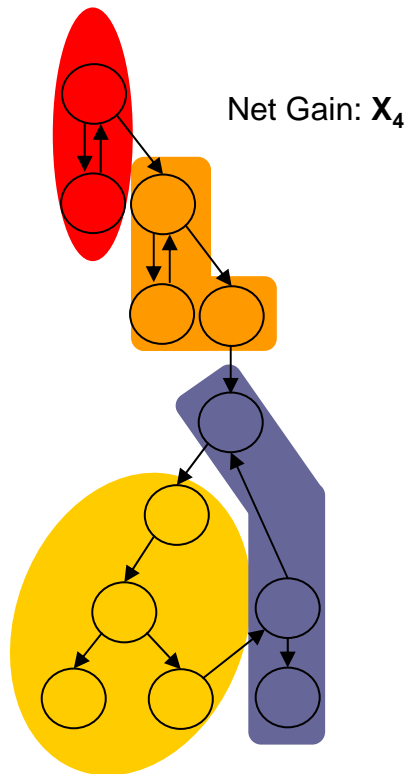
Goal: maximize net performance gain by maximizing performance gain of spatial partitioning and minimizing reconfiguration cost of temporal partitioning

Iterative Solution

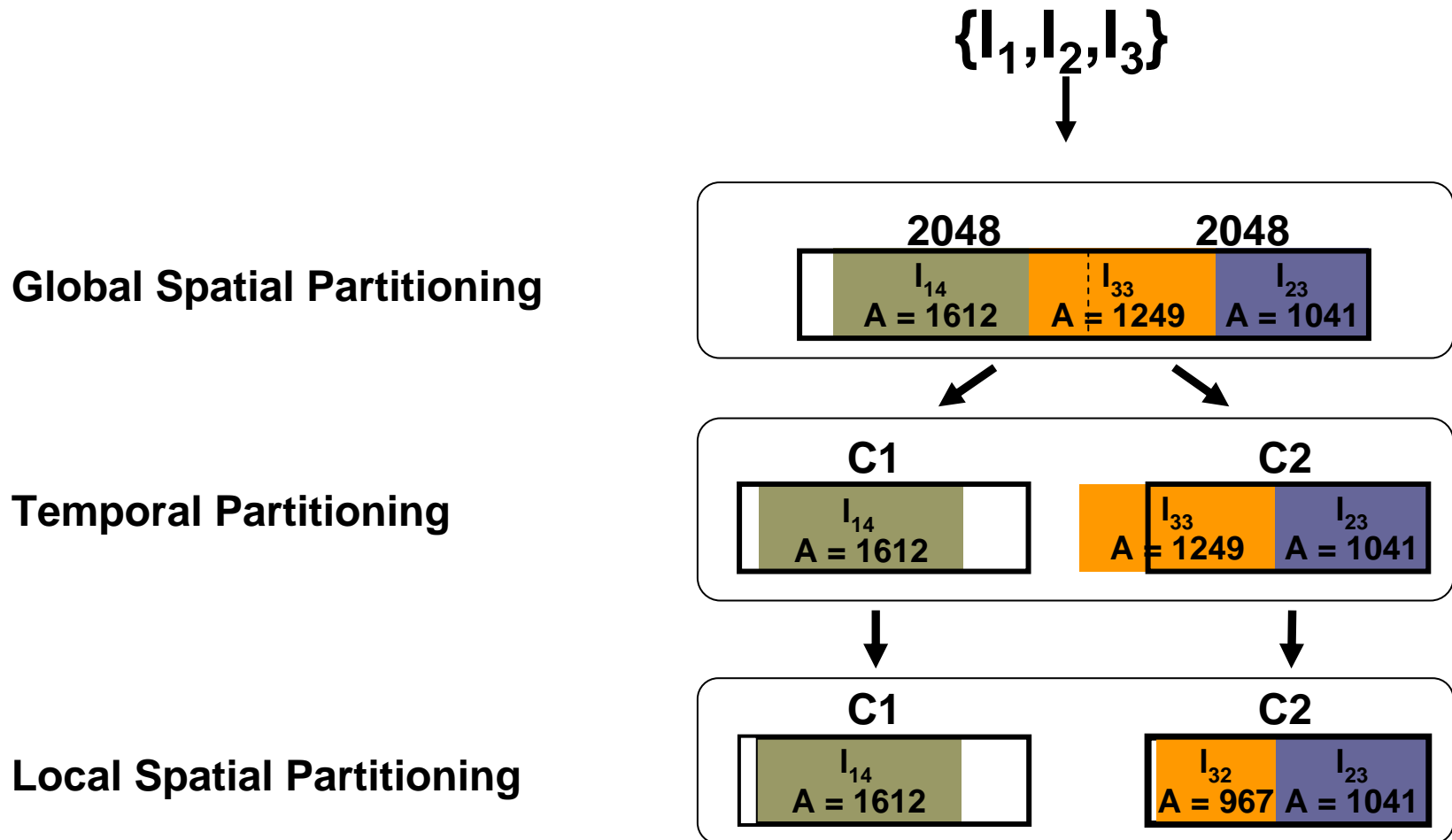
Given K hot loops:

What is the best we can do with only k configurations?

Solution = $\text{Max}(X_1, X_2, \dots, X_k)$

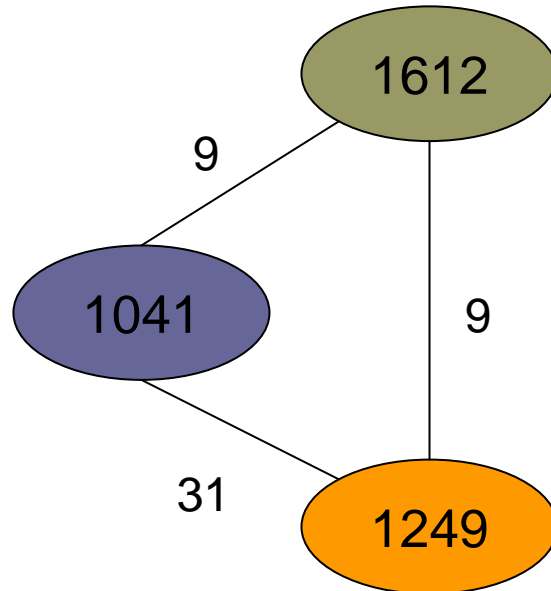


Iterative Step



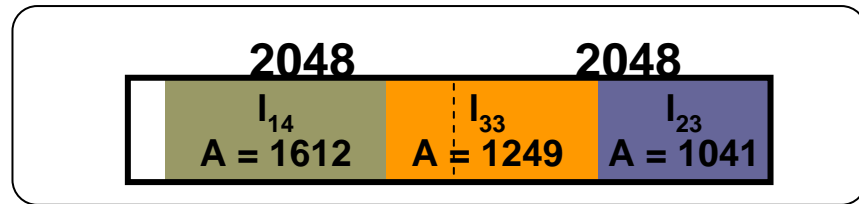
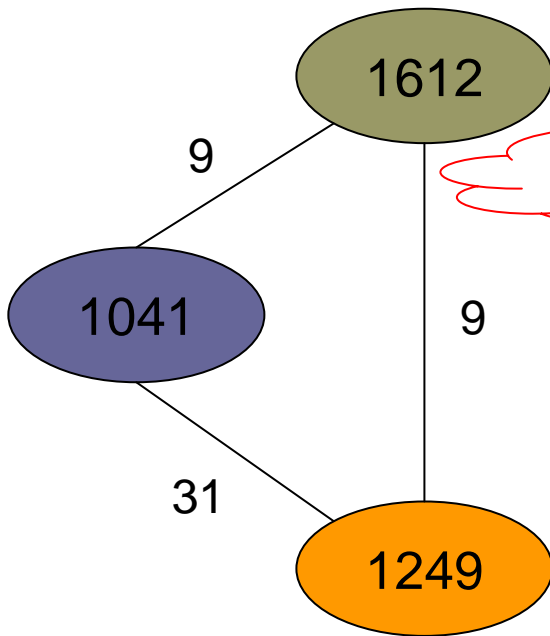
Temporal Partitioning

- Generate a Reconfiguration Cost Graph (RCG) from the loop trace:

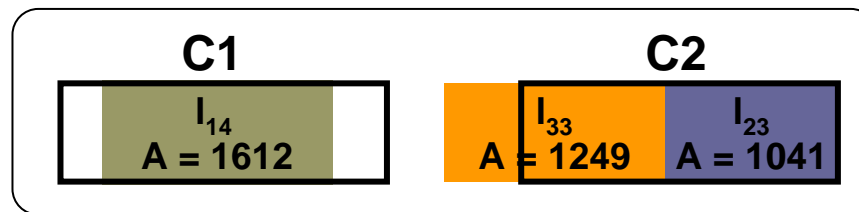
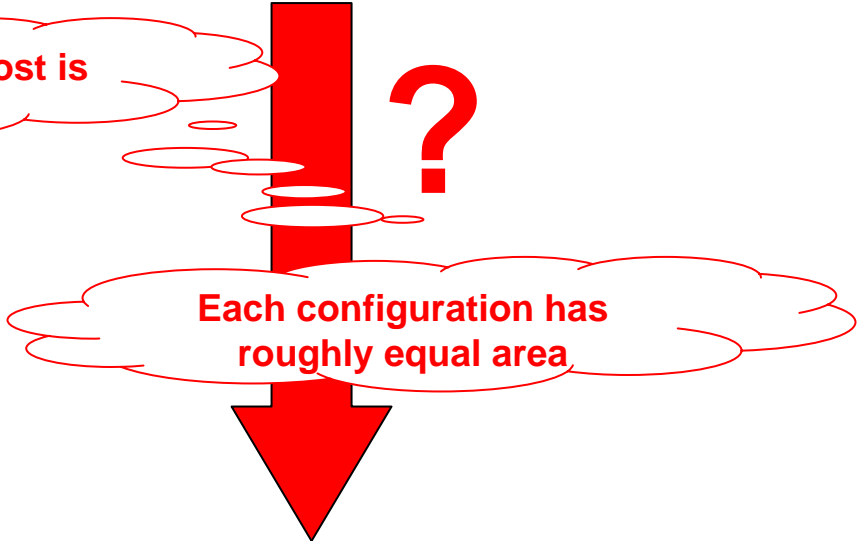


- **Objective:** partition the RCG into k configurations such that the configurations have roughly equal area and the reconfiguration cost is minimized.

Illustration of Temporal Partitioning



Reconfiguration Cost is minimized



K-way Graph Partitioning

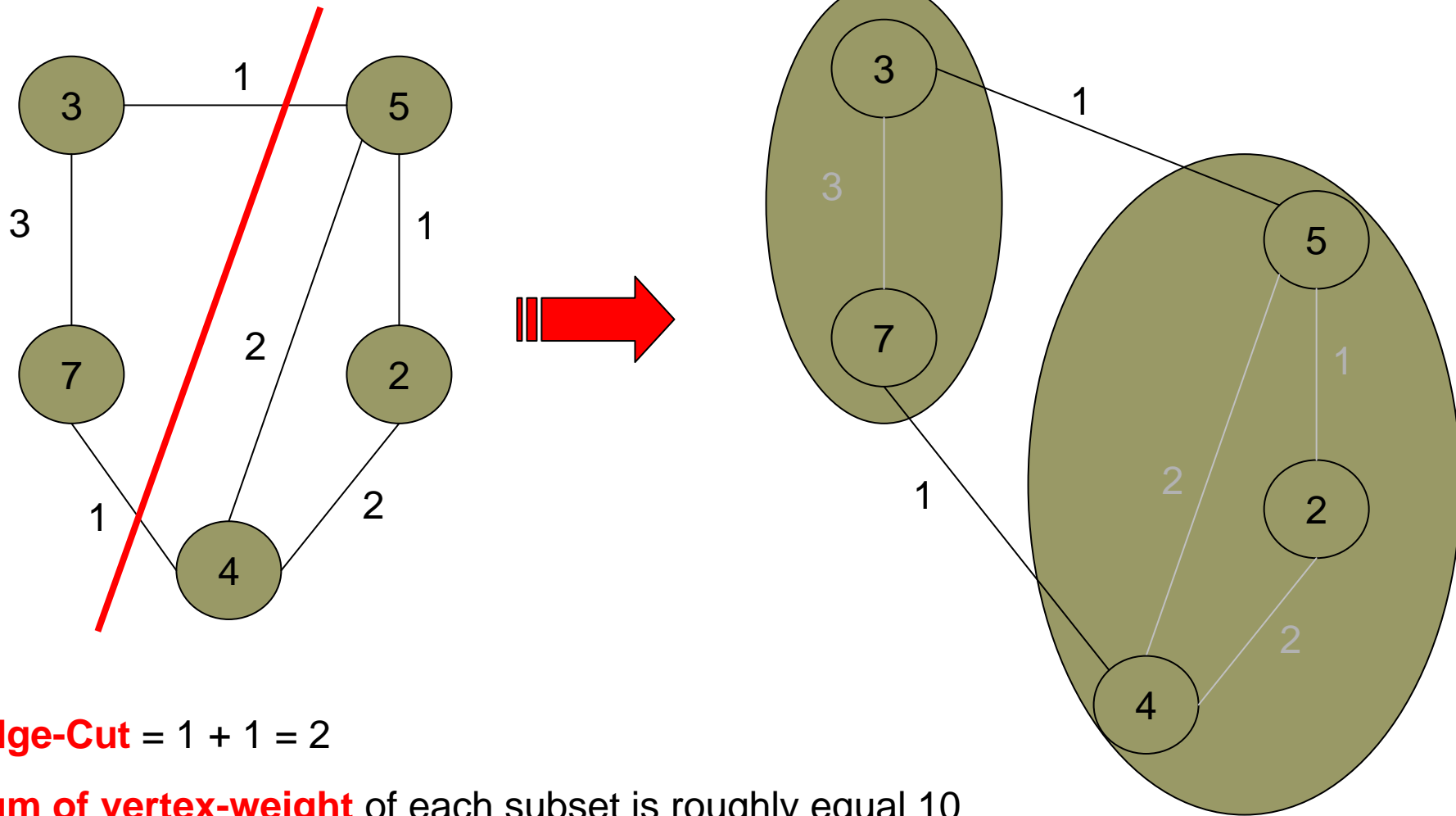
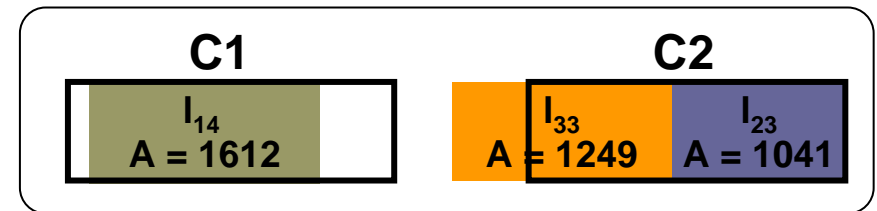
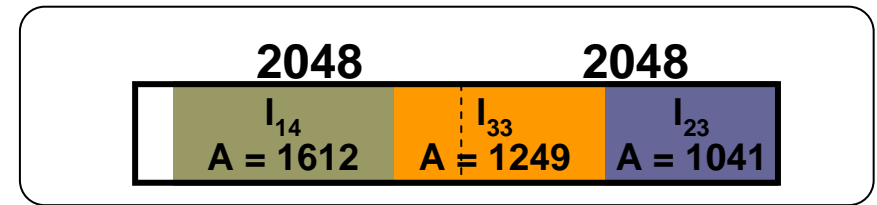
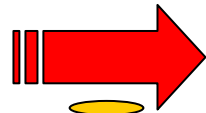
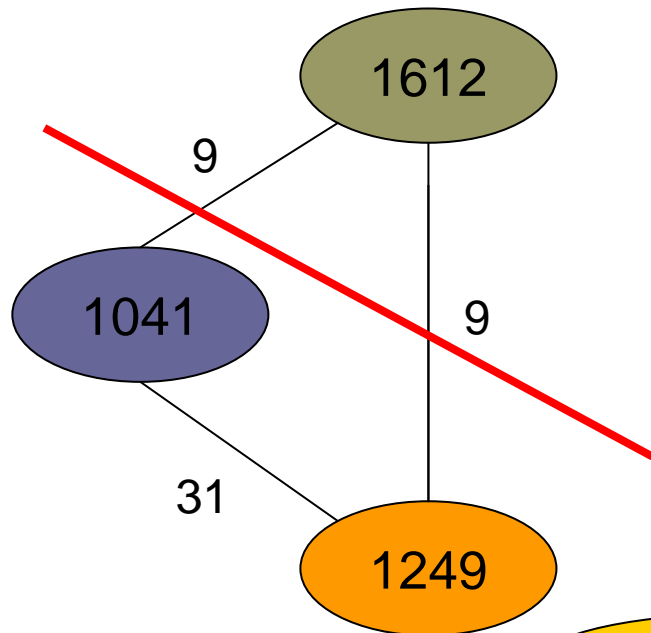


Illustration of Temporal Partitioning



Reconfiguration Cost -> Edge-cut
Area of a configuration -> Sum of
vertex- weight

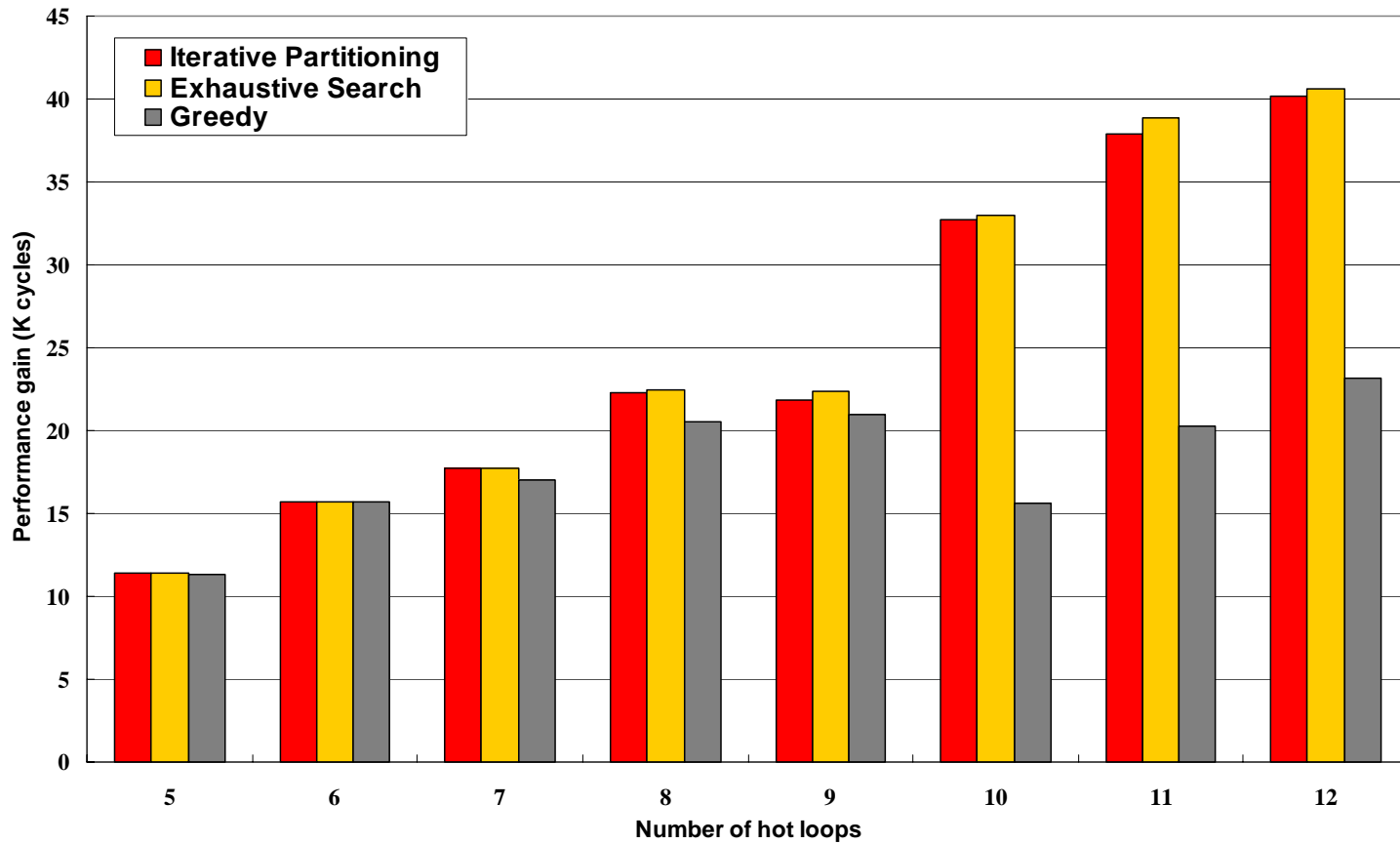
Related Work

- ❑ Custom instruction generation: Atasu 2003, N. Clark 2003, J. Cong 2004, R. Kastner 2002, Arnold 2001
- ❑ Custom instruction selection: Arnold 2001, J. Cong 2004, N. Clark 2003, J. Lee 2002, N. Cheung 2002
- ❑ Major research on temporal partitioning comes from the reconfigurable computing community:
S. Banerjee 2005, M. Kaul 1999, K. Chatha 1999, Purna 1999, Bondalapati 1998, C. Hardnett 2006.
- ❑ Y. Li 2000: considers a single loop in one configuration and does not consider global reconfiguration cost when selecting loops to put in hardware.

Experimental Set-up

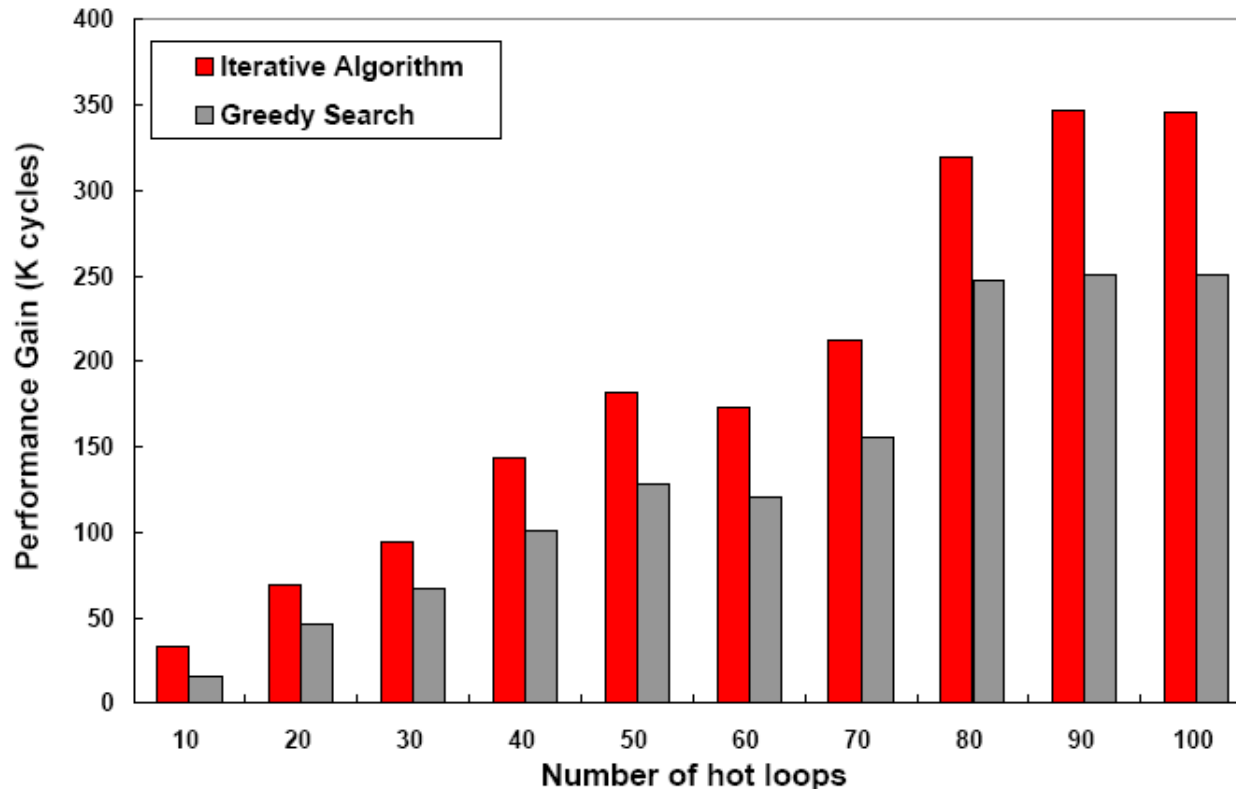
- ❑ The exhaustive search algorithm computes the optimal results by evaluating all possible temporal and spatial partitioning.
- ❑ The greedy search algorithm constructs a solution by building one configuration at a time until no more CIS version can be added without causing a degradation in performance.
- ❑ We perform experiments using synthetic benchmarks as well as a case study (JPEG) application.
- ❑ The experiments were performed on 2.8GHz Pentium 4 machine running Linux FC6.

Result of Synthetic Benchmark



**Comparison among Exhaustive Search, Greedy Search
and Iterative Partitioning**

Result of Synthetic Benchmark



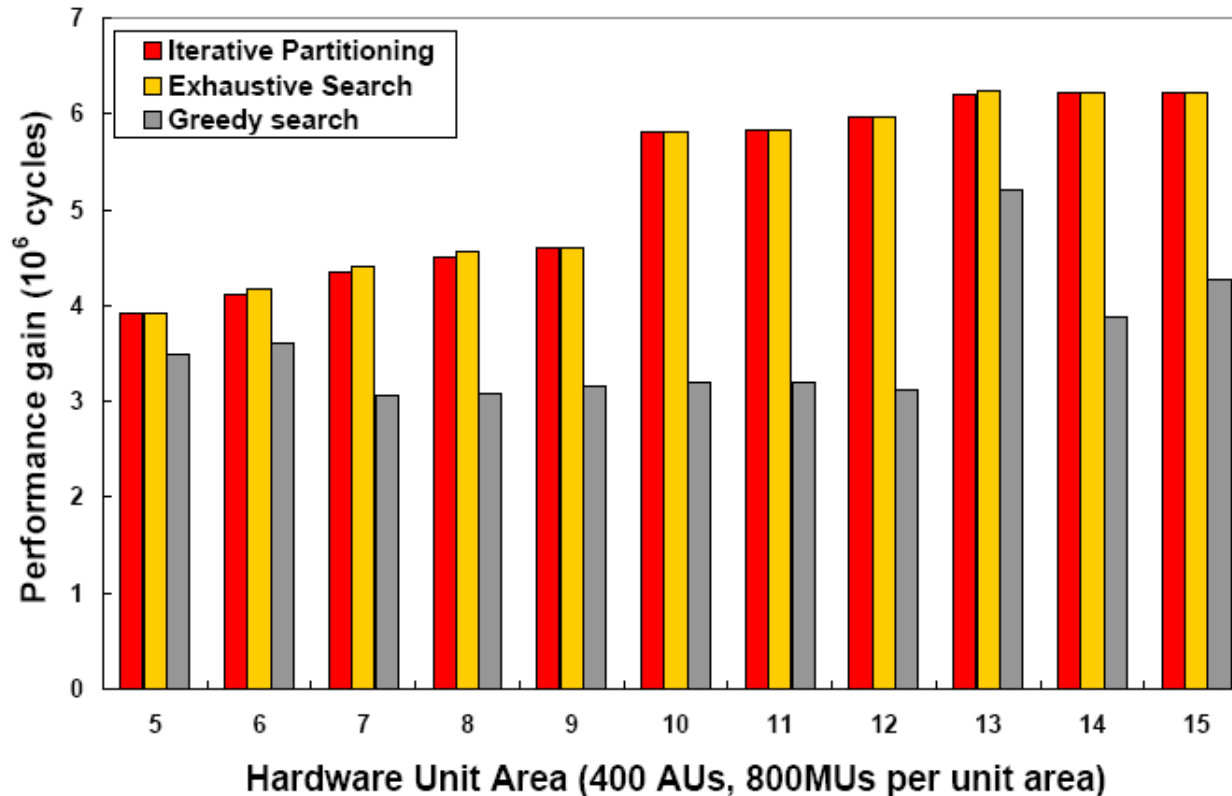
**Relative performance gain of iterative algorithm
compared to greedy search**

Scalability testing

No. of Hot Loops	Exhaustive Search	Greedy Search	Iterative Algorithm
5	0.26	0.01	0.07
6	1.34	0.02	0.07
7	7.84	0.01	0.07
8	43.91	0.01	0.09
9	283.22	0.04	0.07
10	1788.2	0.01	0.11
11	12604.33	0.01	0.13
12	86338.37	0.01	0.15
20	N.A.	0.02	0.48
40	N.A.	0.04	4.3
60	N.A.	0.07	18.25
80	N.A.	0.11	55.61
100	N.A.	0.16	118.76

Running Time (sec)

JPEG Case Study



Comparison among Exhaustive Search, Iterative Partitioning, Greedy Search and 1 Configuration

Conclusions

- Proposed an algorithm to exploit dynamically configurable custom functional units for optimal performance gain.
- The experimental results show that our algorithm is highly scalable while producing optimal or near-optimal performance gain.
- The work can be extended by considering configuration prefetching and partial reconfiguration.

Questions

THANK YOU!