Workload Characterization of a Parallel Video Mining Application on a 16-way SMP Machine

Wenlong Li (Intel China Research Centre)
Eric Li (Intel China Research Centre)
Carole Dulong (Google)
Yen-kuang Chen (MTL, CTG, Intel)
Tao Wang (Intel China Research Centre)
Yimin Zhang (Intel China Research Centre)

Transition to Multi-Core Era

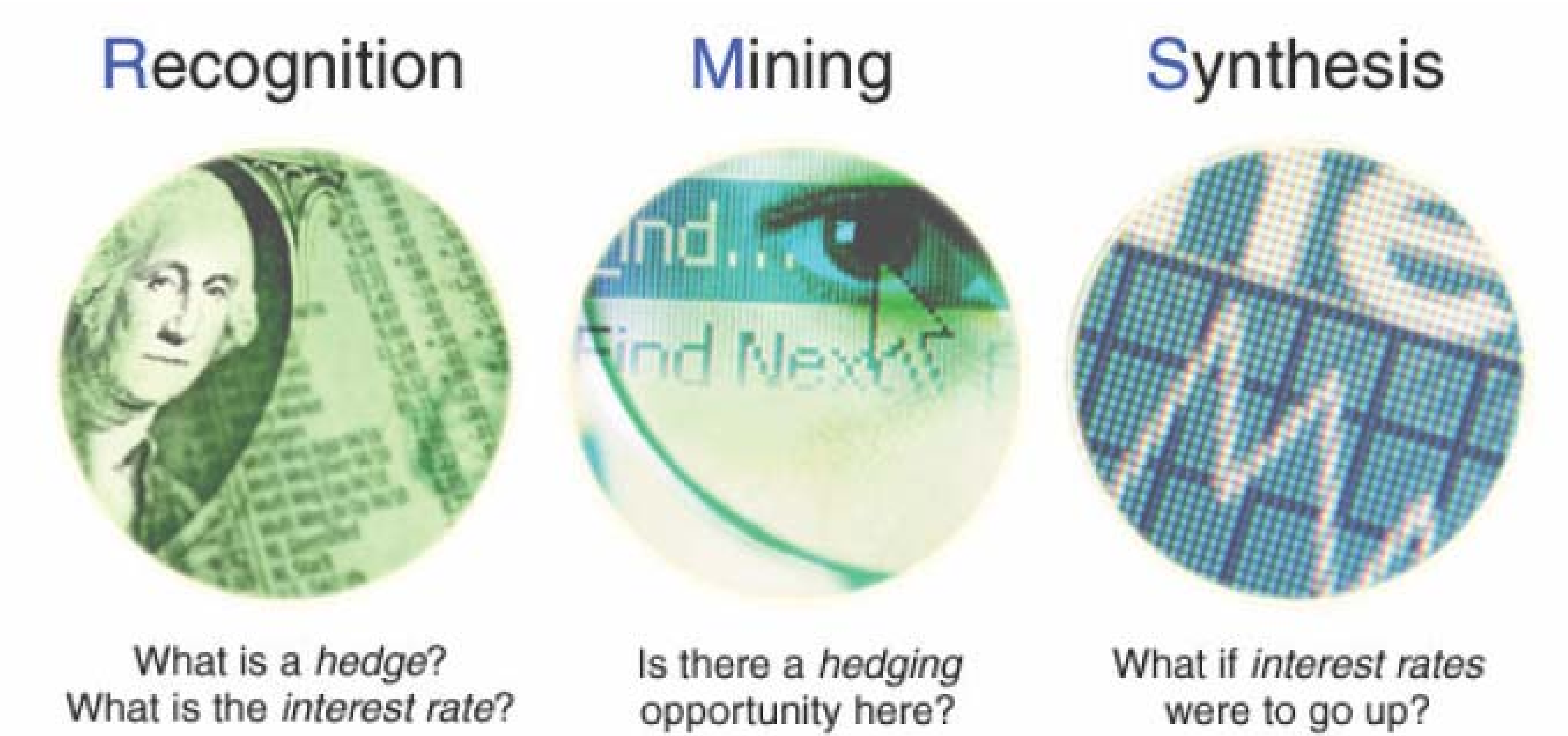
Multi-Core Transition Accelerating

"We notified customers we're pulling in both the desktop and server (launch) of the first quad-core processors into the fourth quarter of this year from the first half of 2007"

"The UltraSPARC T1 processor with CoolThreads technology is the highest-throughput and most eco-responsible processor ever created."

"Azul has been able to pack an industry-leading 24 processor cores on a single-chip, which means that each processor is able to run 24 simultaneous parallel threads"

Emerging 'Killer' Applications (RMS)



Sports video highlight extraction is an example of video mining system

- **Extract highlights from soccer game video**
- **More than 100 million soccer fans in China**

Agenda

Video mining application – soccer highlight detection

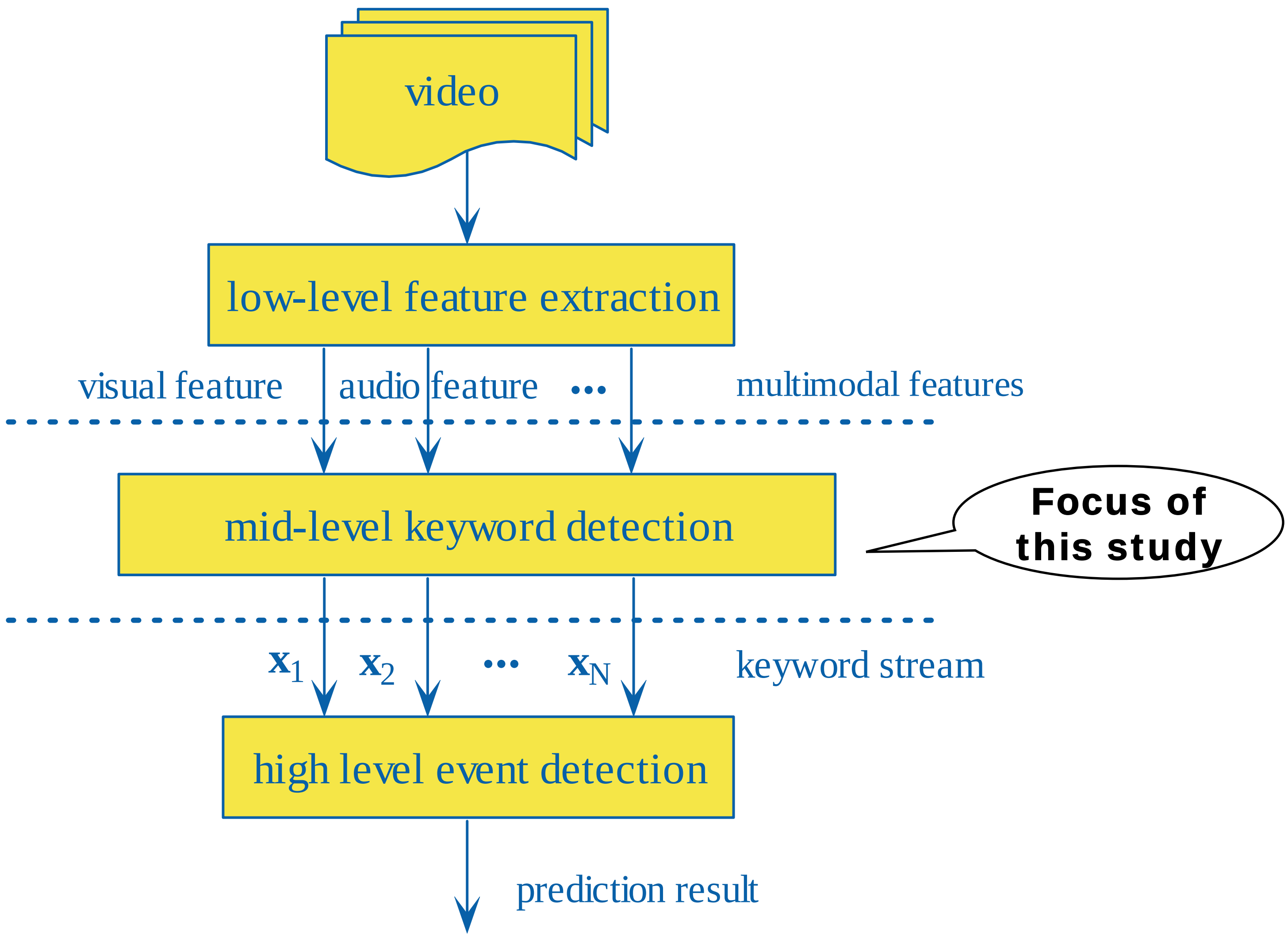
- Framework
- Middle-level keyword detection

Workload optimization and parallelization

Experimental results

Conclusion

Framework

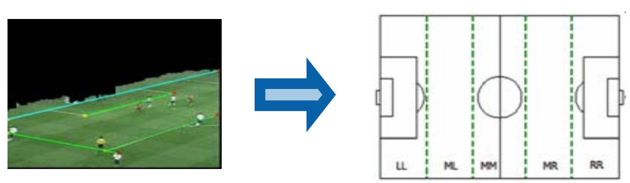

video
low-level feature extraction
visual feature
audio feature
•••
multimodal features
mid-level keyword detection
Focus of this study
$\mathbf{x}_1$
$\mathbf{x}_2$
•••
$\mathbf{x}_N$
keyword stream
high level event detection
prediction result

Middle-level keyword detection

View type detection based on play field size and position

Global, Middle, Close-up

Play field detection based on lines detected by Hough transform



Play Field: 5 Regions

Agenda

Video mining application – soccer highlight detection

Workload optimization and parallelization

- Performance optimization
- Parallel schemes

Experimental results

Conclusion

Performance optimization

Optimization

- SIMD optimization (OpenCV & IPP)
- Sub-expression optimization
- SIMD optimization (Hough line detection)

Performance gain

- SIMD optimization: 3x~4x
- Sub-expression optimization: 24%
- SIMD optimization (Hough line detection): 45%

Parallel schemes

Parallelism exploration

- Task level parallel scheme
- Data level parallel scheme
- Hybrid of task and data parallel scheme

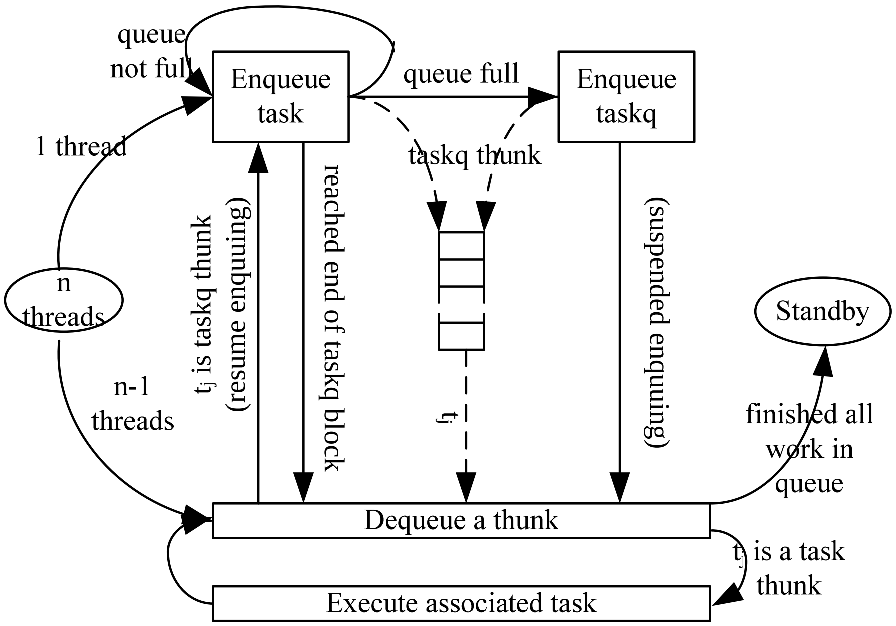
Challenges

- General factors: Load imbalance, Synchronization, etc
- Memory sub-system: Cache optimization, Bus contention mitigation
- Thread scheduling: Exploiting data locality and bandwidth utilization

Task level parallel scheme

This application exhibits the producer-consumer pattern -- a natural fit with OpenMP TaskQ implementation

The master thread is responsible for video decoding, while slave thread performs feature extraction



☞**The ratio between video decoding and feature extraction is important to keep the target system balanced for this scheme**

Data level parallel scheme

Idea

- Split the input video stream into different data chunks
- Perform same computations on each data chunk

Advantage

- Expose more thread level parallelism than task level scheme

Disadvantage

- May experience load imbalance problem

Hybrid of task and data parallel scheme

Combine previous two schemes to form a new parallel scheme

Hybrid central implementation

- Multiple producers (master), multiple consumers (slave)
- Create a central buffer to save tasks produced by master threads
- Fetch tasks from central buffer to perform feature extraction on slave threads
- May incur high synchronization and poor data locality due to the shared buffer

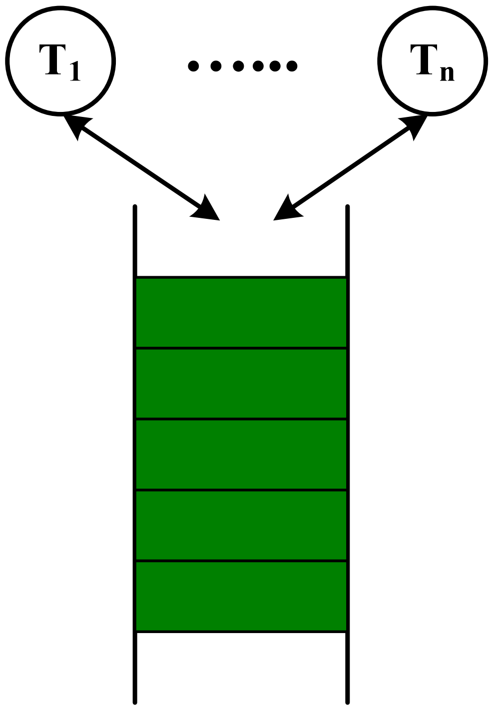
Hybrid distributed implementation

- Create several buffers to save tasks
- Assign each thread to one specific buffer
- May suffer load imbalance problem, but with a good data locality performance

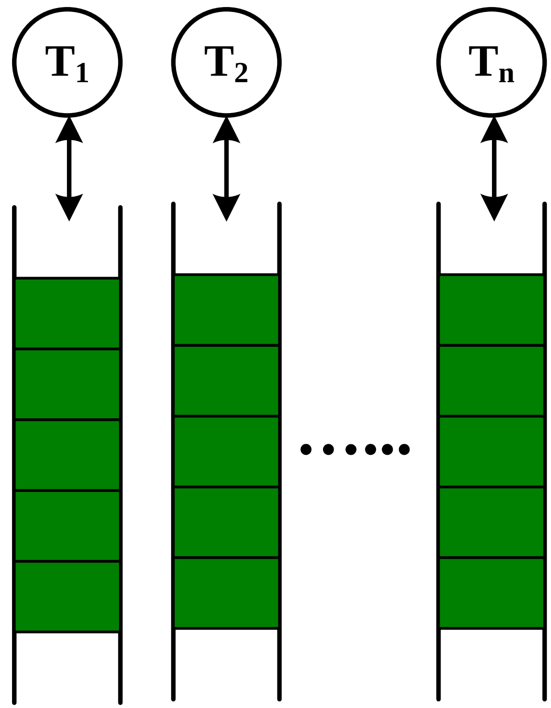
Hybrid distributed implementation with task stealing support

- Same as distributed scheme
- Enable task stealing from neighbor buffer when current buffer is empty
- Good data locality and no load imbalance penalty

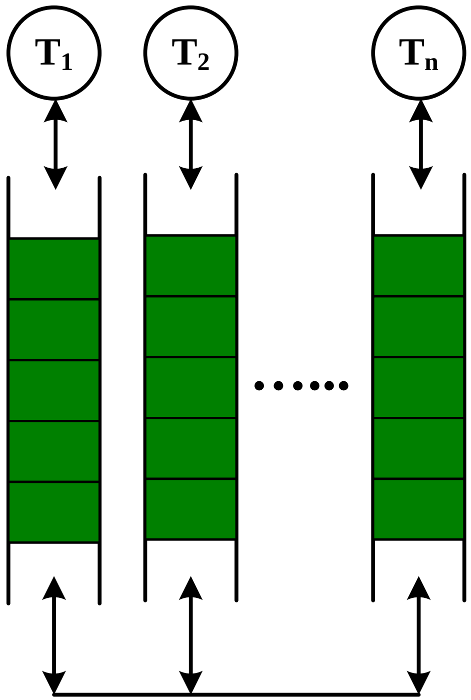
Hybrid of task and data (cont.)



Central scheme



Distributed scheme



Distributed stealing scheme

Agenda

Video mining application – soccer highlight detection

Workload optimization and parallelization

Experimental results

- Speedup performance
- Parallel performance breakdown
- Memory sub-system performance
- Hyper-threading effect
- H/W prefetching effect
- Thread scheduling effect

Conclusion

Experimental setup

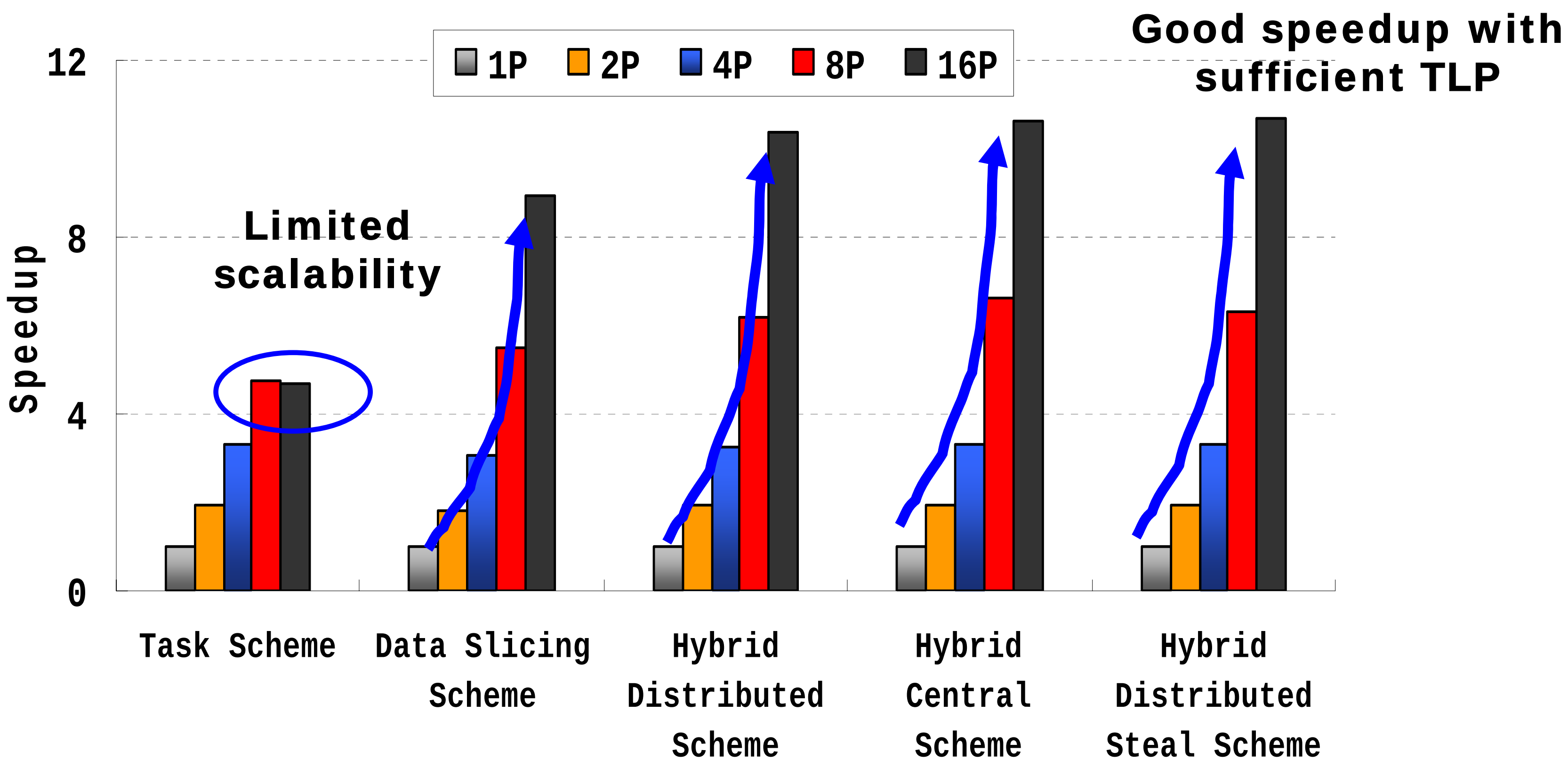
Hardware (Unisys machine)

- 16 Way Intel Xeon SMP machine
- Each processor has 8KB L1 D-cache, 512KB L2 cache, 4MB L3 cache
- Each four processor forms a cluster and shares a 32MB LLC
- Crossbar is used for interconnect (3.2GB/s)

Programming tool

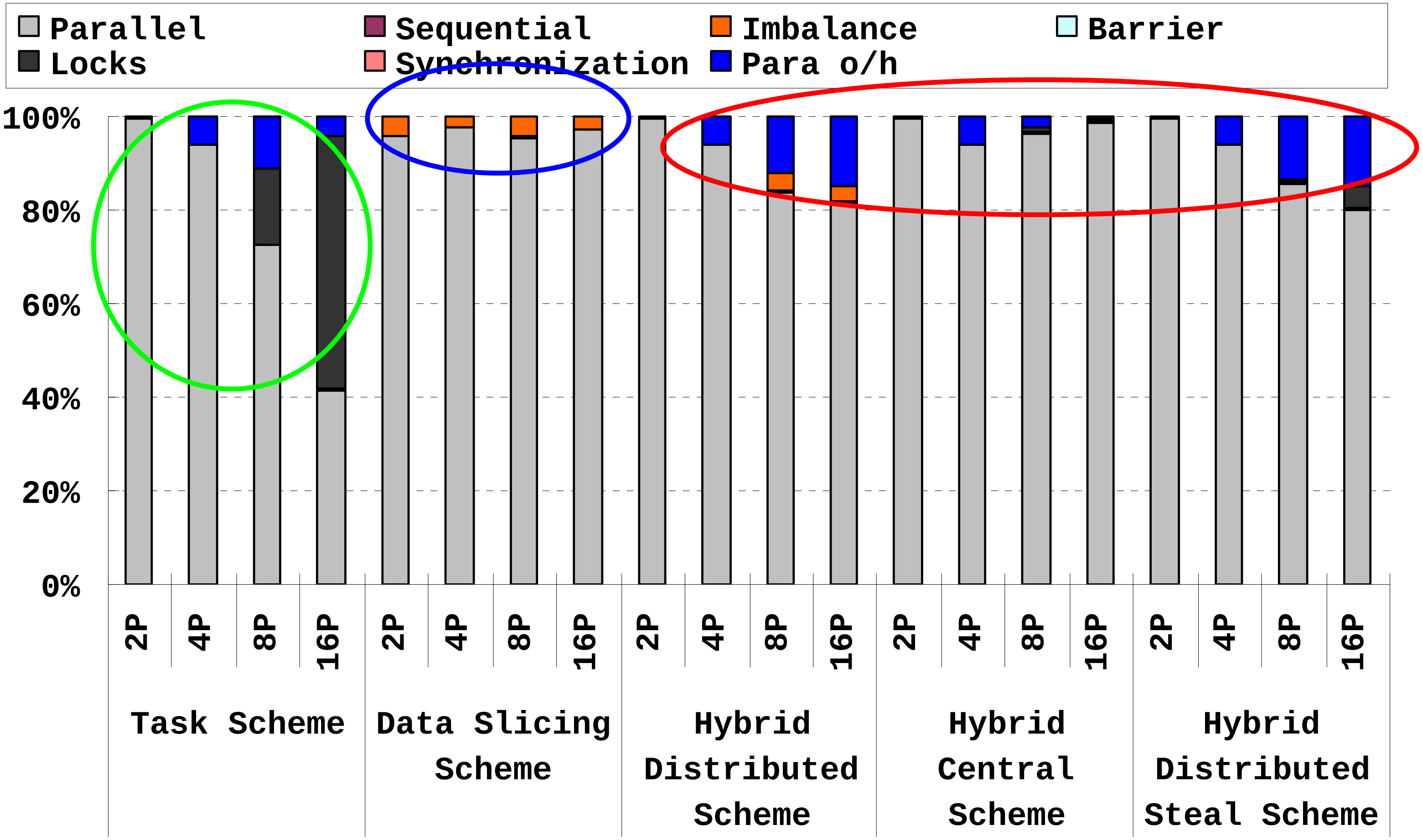
- OpenMP is used to parallelize this application
- Intel IPP and OpenCV are used for SIMD optimization
- Intel VTune and Thread Profiler are used to collect performance metrics

Speedup comparison



- ☞Hybrid scheme achieves the best performance
- ☞Comparable speedup for three hybrid scheme on 16 processors
- ☞ Hybrid distributed with stealing scheme performs best among these three schemes on 16 processors

Parallel performance metrics



- Task scheme suffers from high synchronization overhead due to limited parallelism
- Data parallel scheme incurs load imbalance problem
- Hybrid parallel schemes perform better with low performance limiting metrics

Memory sub-system performance

Cache miss rate remains flat as processor number increases since each processor executes similar computations

Proc Num	1P	2P	4P	8P	16P
L1 Miss Rate (%)	8	8	8	8	8
L2 Miss Rate (%)	18	18	18	18	18
L3 Miss Rate (%)	9	8	9	9	9
Bus Utilization Rate (%)	6	10	19	36	76
L1 Miss per K Inst	28	28	29	29	29
L2 Miss per K Inst	5.1	5.2	5.1	5.2	5.2
L3 Miss per K Inst	0.46	0.46	0.45	0.47	0.46

☞ **Increase linearly as processor number increases**

☞**Saturated bus utilization is the likely cause of bottlenecks for achieving perfect scalability performance**

The effect of Hyper-Threading

Conduct experiment to example the effect of Hyper-Threading

C1: Run 4 threads on 4 physical processors with HT off (4 physical processors)

C2: Run 8 threads on 4 physical processors with HT on (8 logical processors)

C3: Run 4 threads on 2 physical processors with HT on (4 logical processors)

C2 is 8% slower than C1, and C3 is 40% slower than C1, and C2 is 30% faster than C3.

C1 faster than C2 means HT hurts performance for this workload

C2 only has 30% more performance than C3 while it has double number of processors as C3

The degraded HT performance is due to the high contention on the shared memory hierarchy

The effect of hardware prefetching

This application exhibits the streaming constant stride data access pattern with high data spatial locality

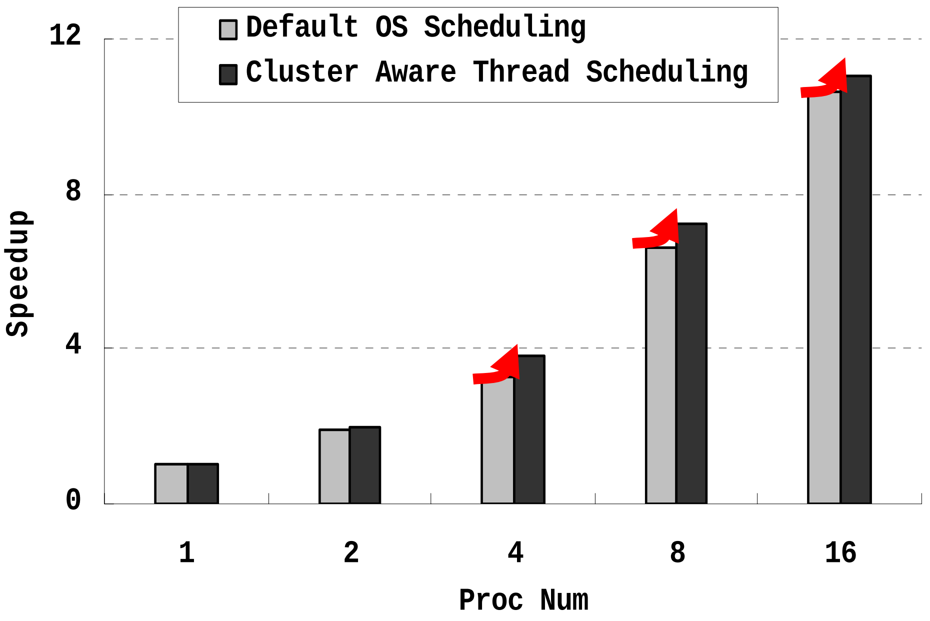
Hardware prefetching is expected to play an important role in delivering the final performance

Proc Num	Prefetch disabled		Prefetch enabled		Gain (%)
	Time(s)	Speedup	Time(s)	Speedup	
1P	778	1.0	681	1.0	12.5
2P	443	1.8	355	1.9	19.7
4P	286	2.7	207	3.3	27.8
8P	143	5.4	102	6.6	28.5
16P	87	8.9	64	10.6	26.3

The effect of thread scheduling

On cluster organized Unisys machine

- Each cluster has one memory bus
- Schedule threads to different clusters first to utilize the aggregated interconnect bandwidth
- Schedule closely coupled threads on same cluster to improve data locality of LLC
- Term as cluster aware thread scheduling



☞ **By considering both bandwidth and data sharing, parallel performance is further improved**

Conclusion

This is a bandwidth limited application on our SMP machine

- Projected bandwidth requirement is 3.1GB/s for 16 cores
- Projected bandwidth requirement is 6.2GB/s for 32 cores

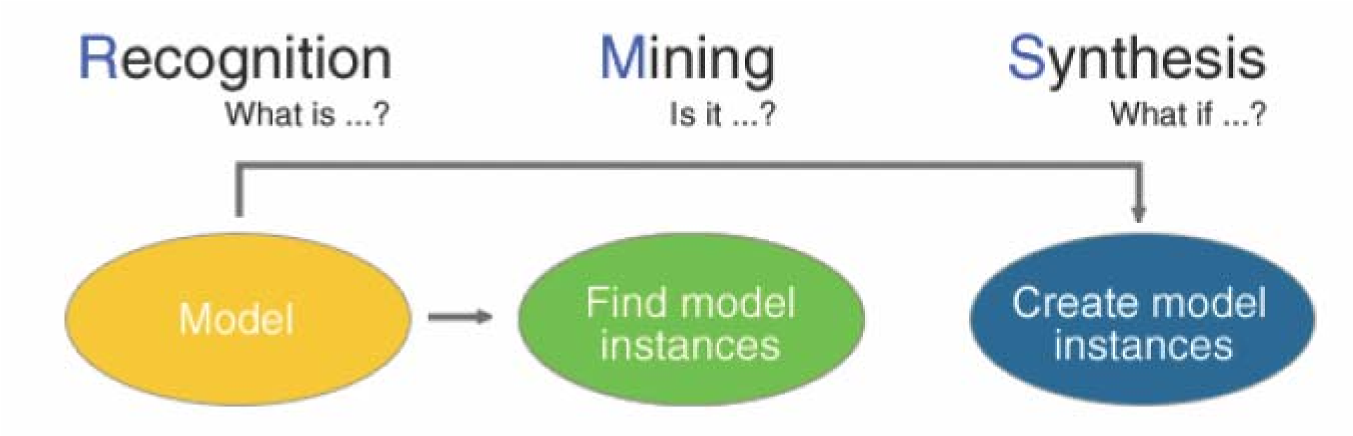
Hybrid parallel scheme achieves the best performance

- 10.6 (hybrid) vs 8.95 (data) vs 4.72 (task)

This application would scale well on MCA with high bandwidth

- Low parallel overhead
- Low sequential percentage
- Low imbalance problem
- Low synchronization overhead
- Simulation on MCA simulator verifies this point

Questions?



This diagram shows how RMS can be used to create a model, find instances of that model, and predict what a model instance might be like where there isn't one