Bound-Oriented Parallel Pruning Approaches for Efficient Resource Constrained Scheduling of High-Level Synthesis

Mingsong Chen, Lei Zhou, Geguang Pu and Jifeng HE

Shanghai Key Lab of Trustworthy Computing

East China Normal University, China

October 1, 2013

Outline

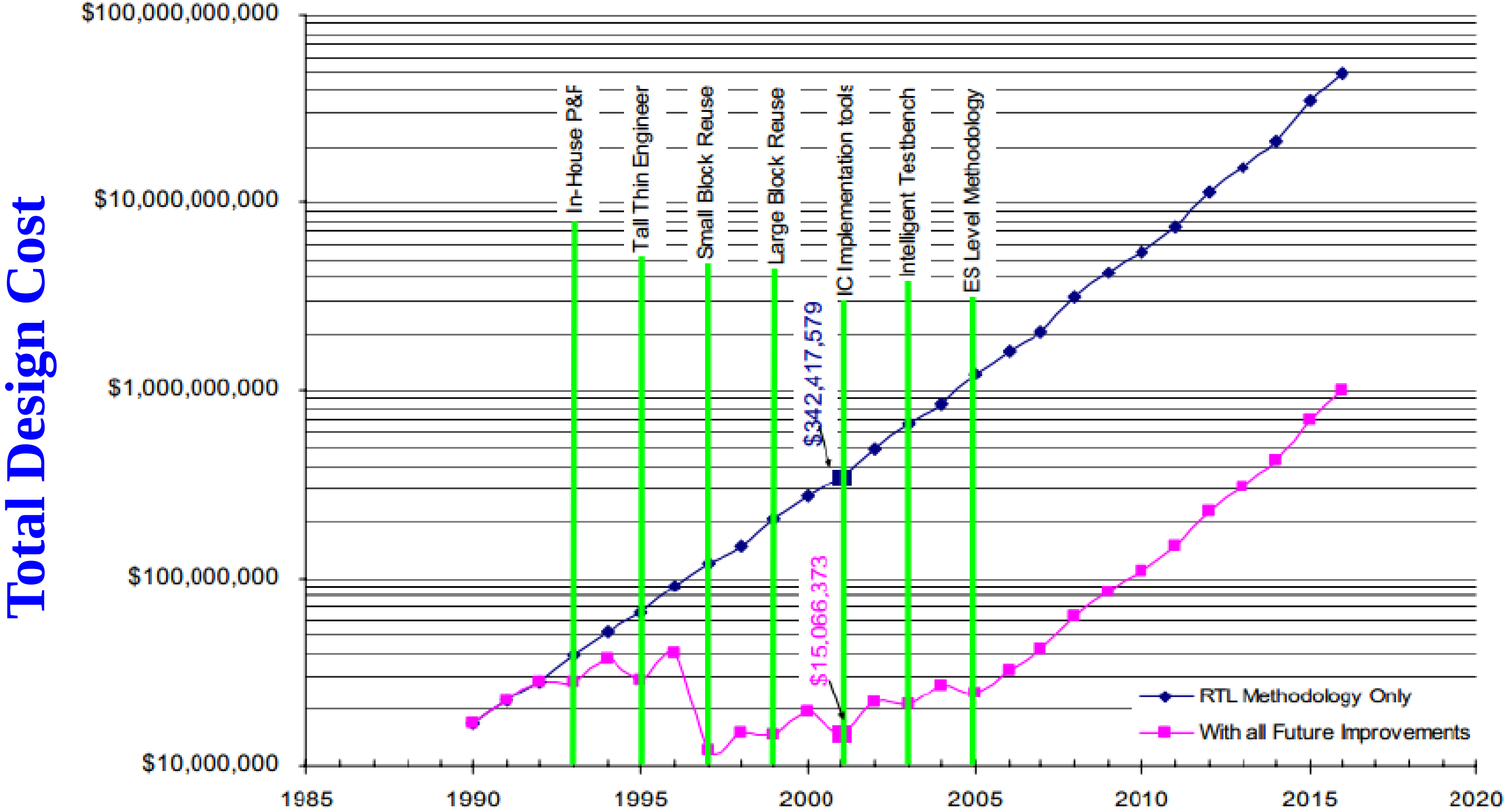
- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion

Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion

SoC Design Cost Model

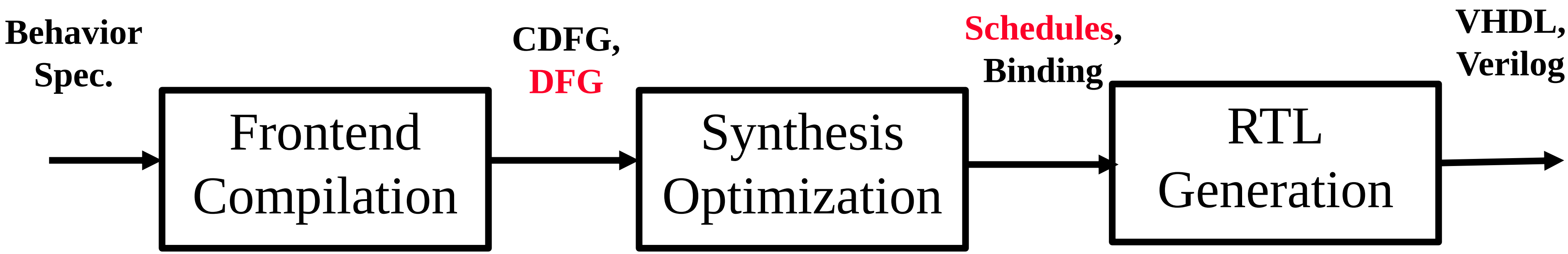
Big Savings by using ESL Methodology



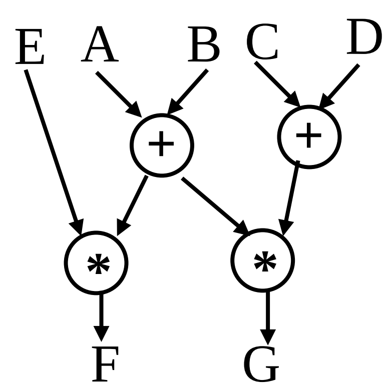
Rising cost of IC design and effect of CAD tools
(Courtesy: Andrew Kahng, UCSD and SRC)

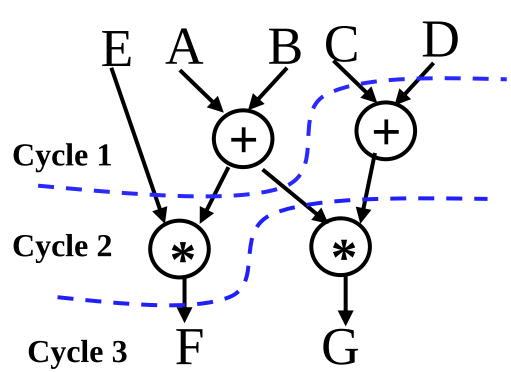
High Level Synthesis

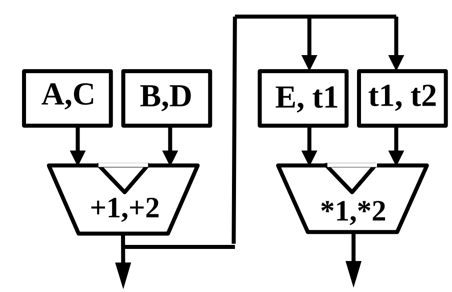
- **Convert ESL specifications to RTL implementations, and satisfy the design constraints.**
  - **Input:** Behavior specifications (C, SystemC, etc.), and design constraints (delay, power, area, etc.)
  - **Output:** RTL implementations (datapath, controller)




```
int Sample(){
int A,B,C,D,E,F,G ;
Read(A, B, C, D, E);
F = E * (A + B);
G = (A + B) * (C + D);
……
}
```

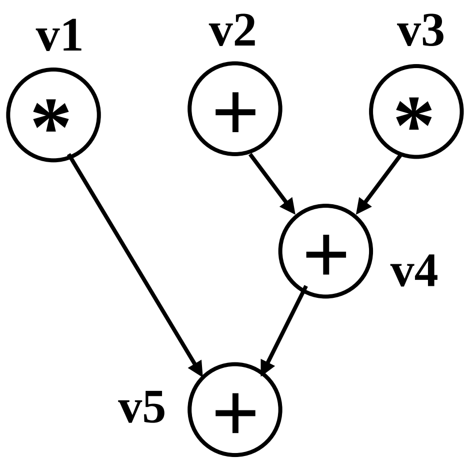




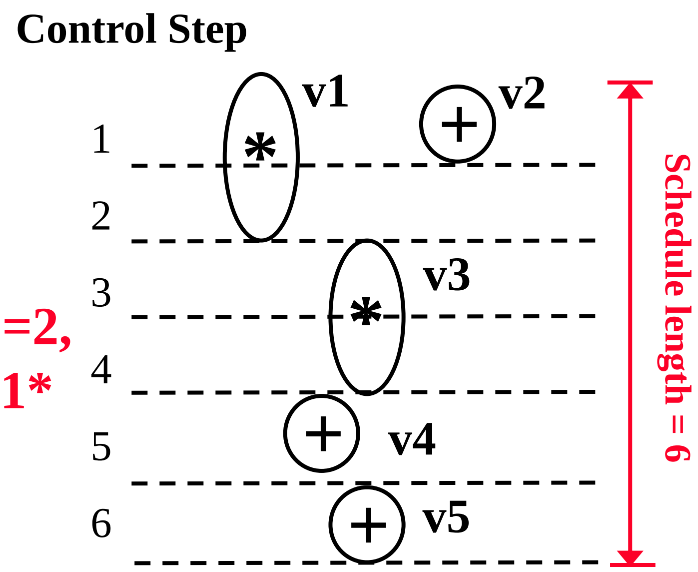


Resource Constrained Scheduling

- **Various resource constraints (e.g., functional units, power, …).**
- **Scheduling is a mapping of operations to control steps**
  - Given a DFG and a set of resource constraints, RCS tries to find a (optimal) schedule with minimum overall control steps.



Constraints:
Delay(+)=1, Delay(*)=2,
functional units: 1+, 1*



RCS is NP-Complete. RCS should take care of
1) Operation precedence. 2) Resource sharing constraints

Basic Solutions

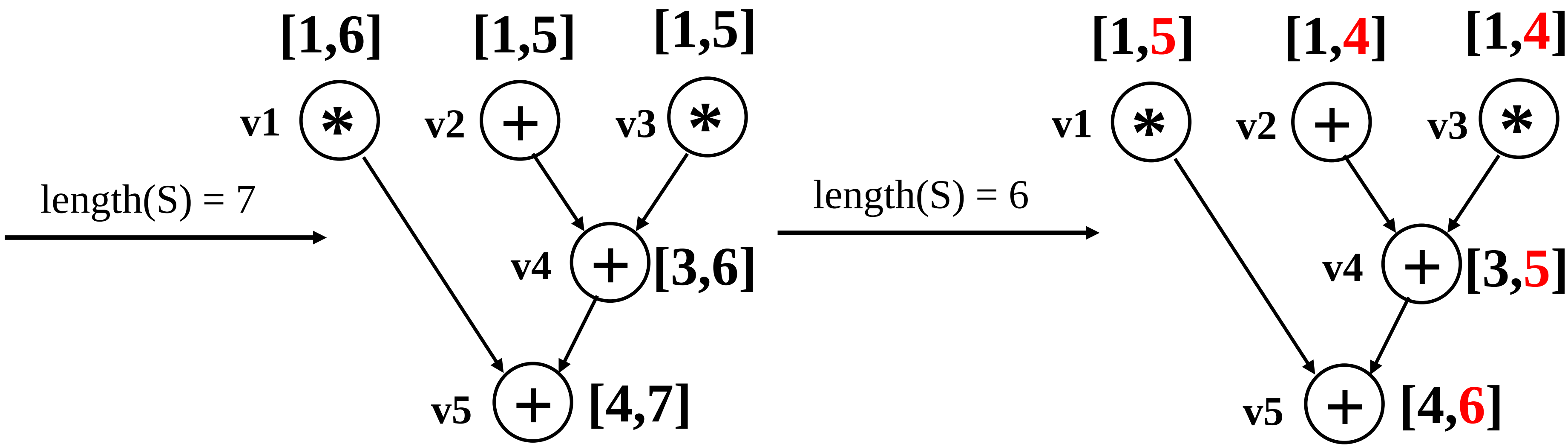
- **Non-optimal heuristics**
  - **Force Directed Scheduling**
  - **List scheduling**
    - ✓ **Pros: Fast to get near-optimal results**
    - ✓ **Cons: schedules may not be tight**
- **Optimal approaches**
  - **Integer linear programming (sequential, parallel)**
    - ✓ **Pros: easy modeling**
    - ✓ **Cons: scalability, cannot handle non-integer time**
  - **Branch-and-bound**
    - ✓ **Pros: can prune the fruitless search space efficiently**
    - ✓ **Cons: few of them support parallel HLS specifically**

Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion

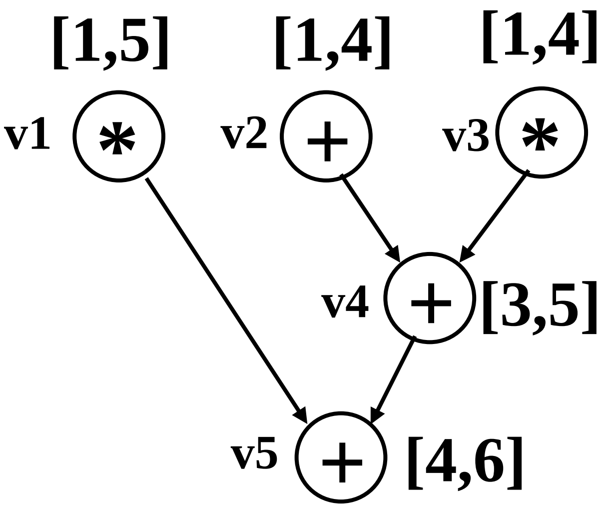
Graph-Based Notations

- [ASAP, ALAP] intervals indicate the earliest and latest start time of operations
- ASAP assumes unlimited resources
  - $ASAP(op_i) = CP(G_{pre}(op_i)) - delay(op_i) + 1$
- ALAP needs to find a feasible schedule S first
  - $ALAP(op_i) = length(S) - CP(G(op_i)) + 1$
  - Update ALAP when obtaining a new better schedule

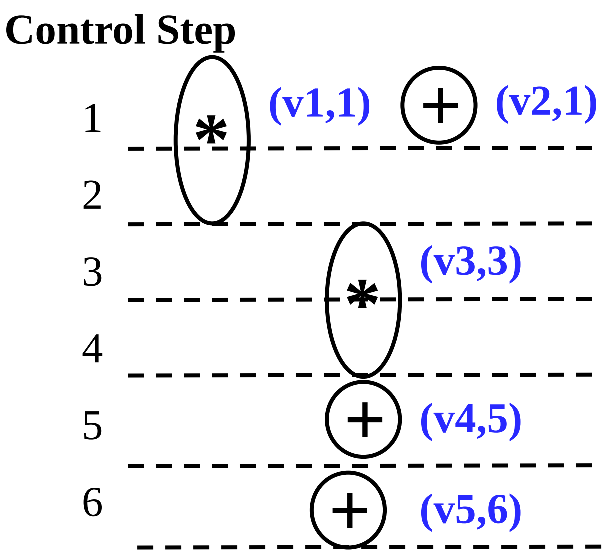


Scheduling Using [ASAP, ALAP]

- A **schedule** is a binary relation of operations and corresponding dispatching control steps
  - E.g., {(v1, 1), (v2, 1), (v3, 3), (v4, 5), (v5, 6)}



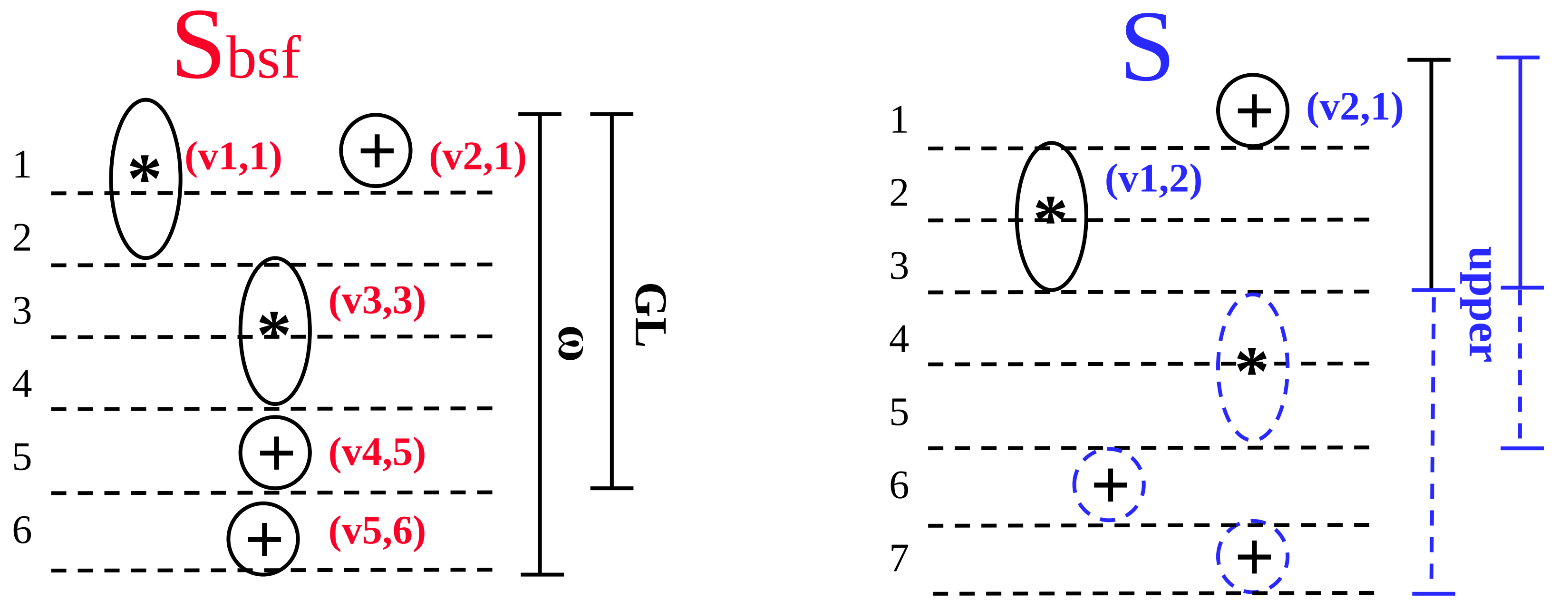
Constraints:
Delay(+)=1,
Delay(*)=2,
1+, 1*



- Based on [ASAP, ALAP], naively enumerating all the possibilities can be extremely time consuming
  - The operations are enumerated in a specific order
  - Each operation is enumerated from ASAP to ALAP

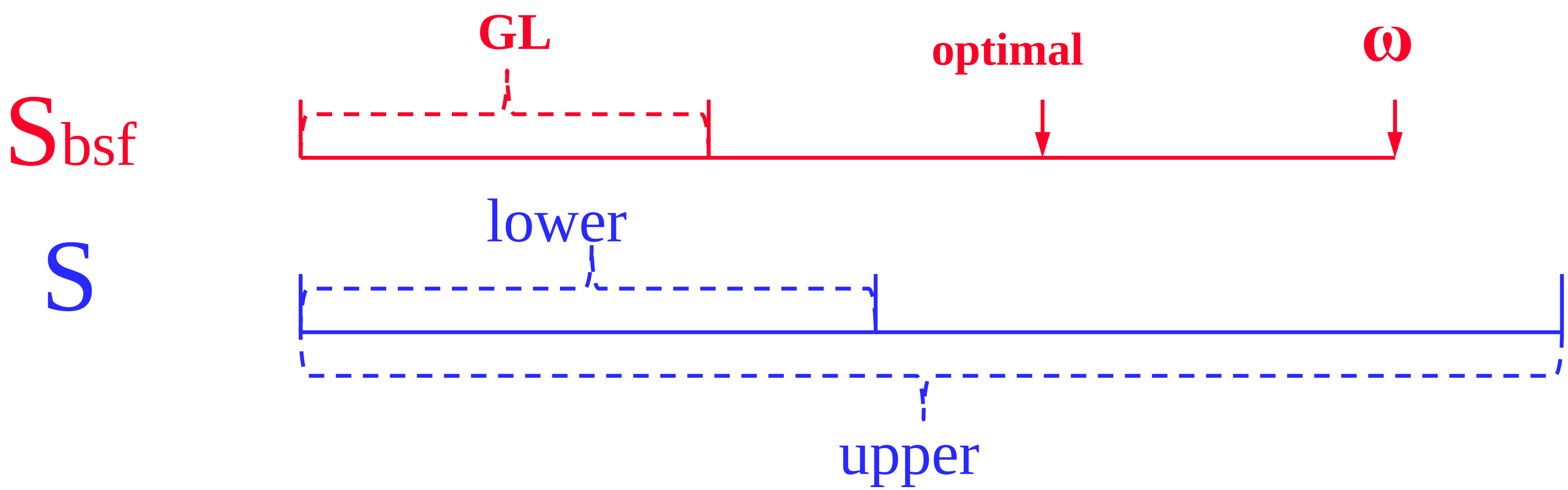
Branch and Bound Style RCS (BULB)

- BULB tries to prune fruitless enumerations.
- B&B approach keeps two data structure regarding bound information.
  - S_{bsf} , best complete schedule searched so far
  - S, current incomplete schedule



Pruning in BULB

- **Pruning [lower > ω]**
- **Termination [GL== ω or fully explored]**
- **Substitution [if(upper < ω) ω = upper]**



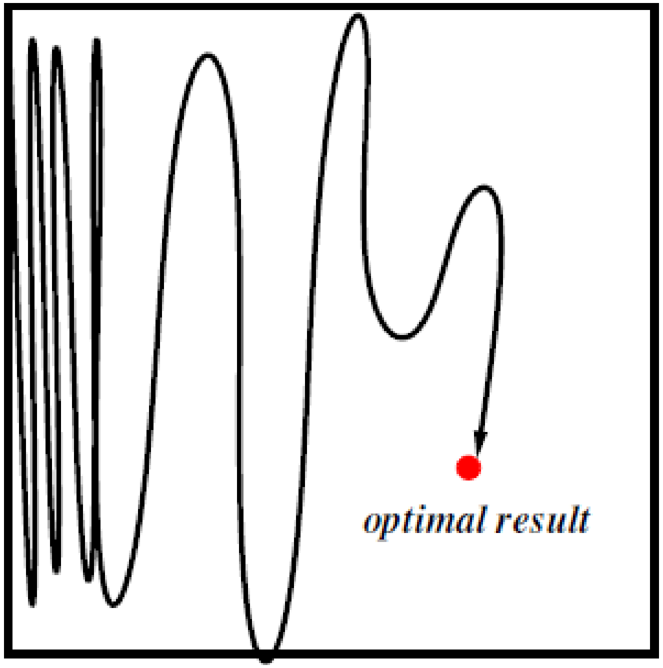
ω plays an important role in B&B approaches. A wise use of ω can
- enable the fast pruning of inferior schedules during RCS;
- tighten the [ASAP, ALAP] intervals, i.e., search space.

Outline

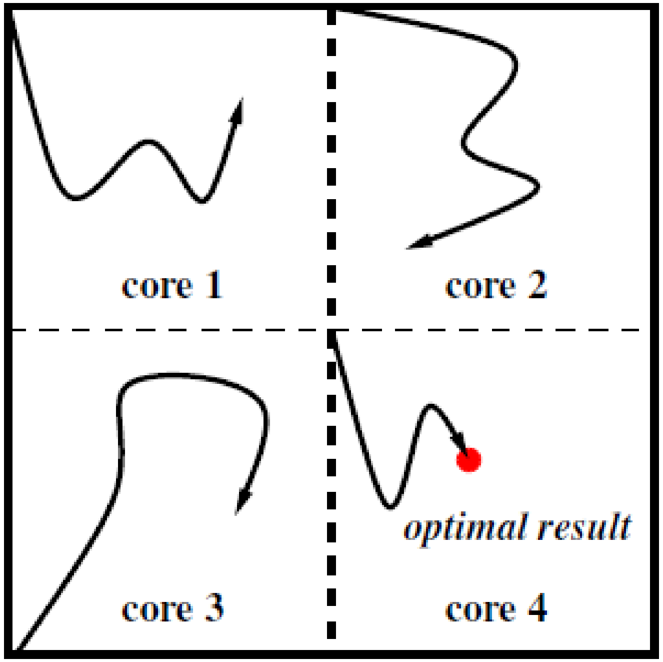
- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion

Search Space Partitioning

- **The search space can be calculated using the cartesian product of [ASAP, ALAP] intervals.**
- **If no better schedule is found, RCS can be easily *stuck-at-local-search*, i.e., be trapped in the deep recursive search.**

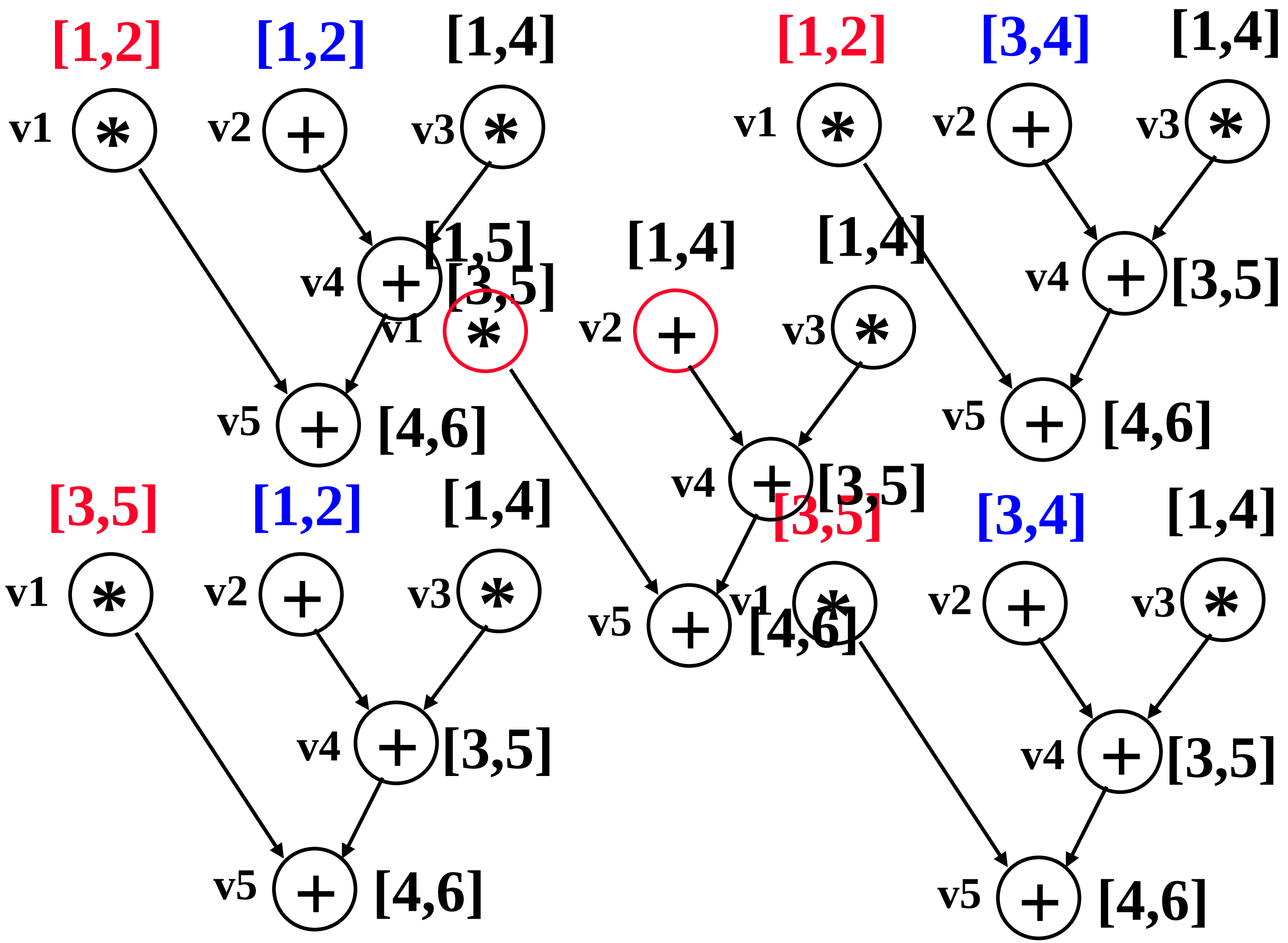


a) Without partitioning



b) With 4 partitions

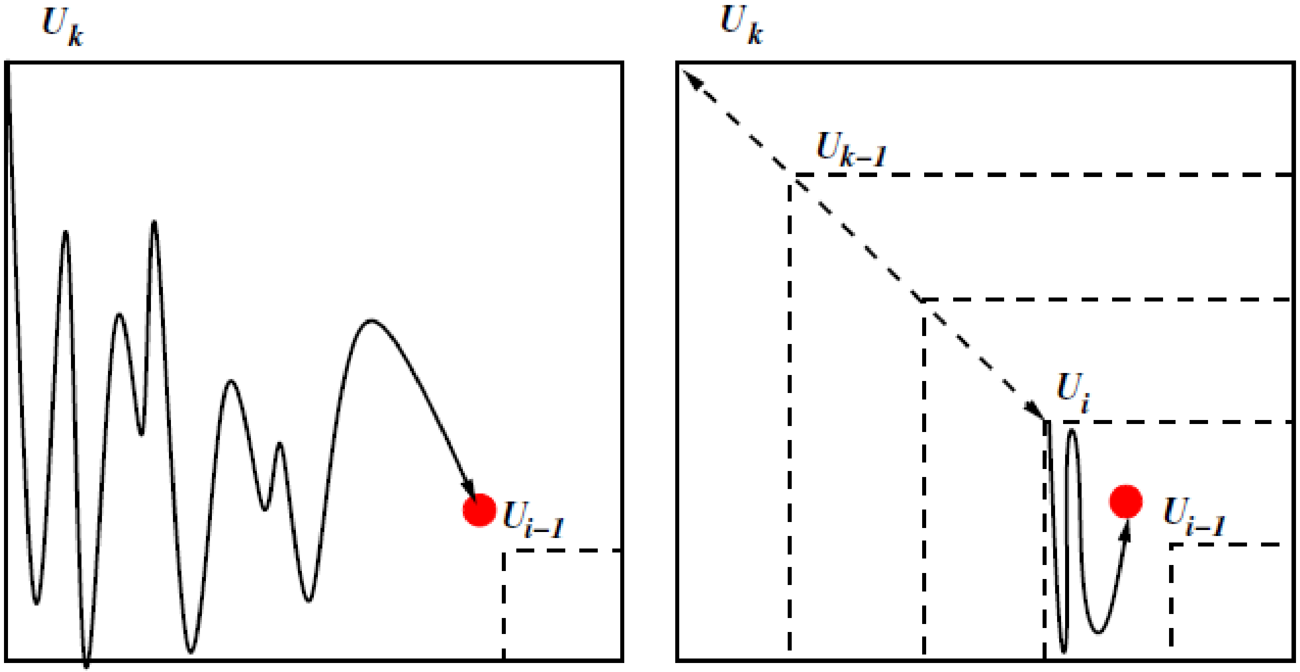
Search Space Partitioning



Termination condition:
1) ω = GL; or 2) all the sub-search finishes.

Static Upper Bound Speculation

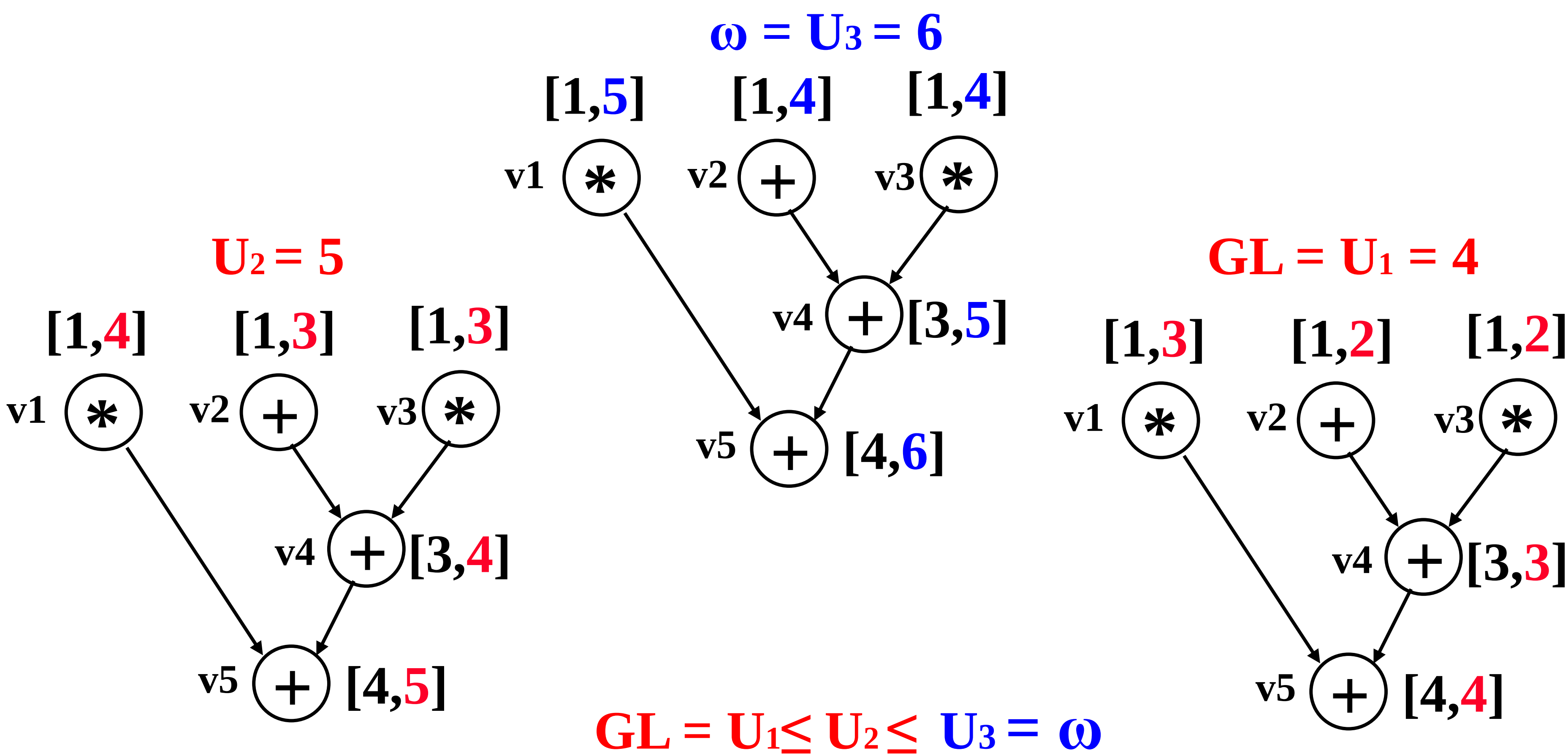
- In BULB approach, the tightest initial **ω** can achieve the best RCS time.
- However, it is hard to achieve such a tightest estimation on a single-core platform .



a) RCS search without the speculation b) RCS search with the speculation

- If there are k cores, the upper bound will be speculated with lengths U_1, U_2, …, U_k where **GL**$=U_1<U_2<\ldots<U_k=$ **ω**

Static Upper Bound Speculation

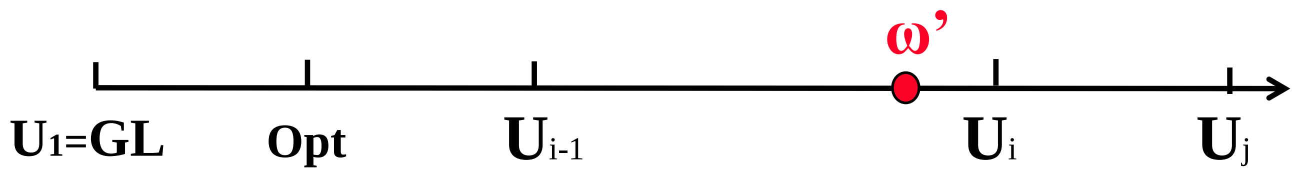


Termination condition:

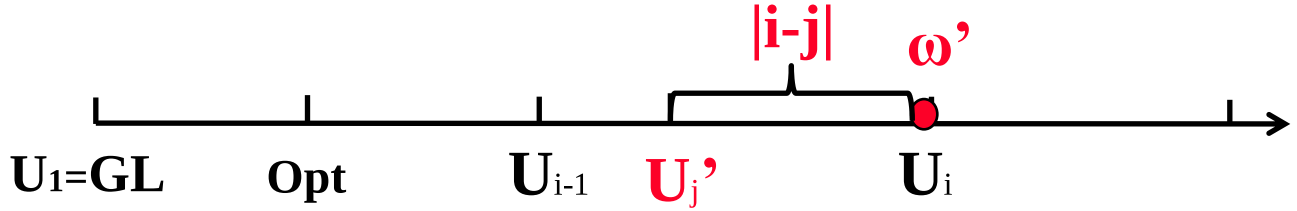
1) ω = GL; or
2) Some sub-task finishes and finds one feasible schedule.

Dynamic Upper Bound Speculation

- Assume that $GL=U_1<U_2< \dots <U_k=\omega$ are *k* upper-bound speculations.
- When a sub-search task find a new ω' such that $U_{i-1}<\omega'<U_i$ $(k>i>1)$. The speculation on U_j $(j>i)$ becomes useless.

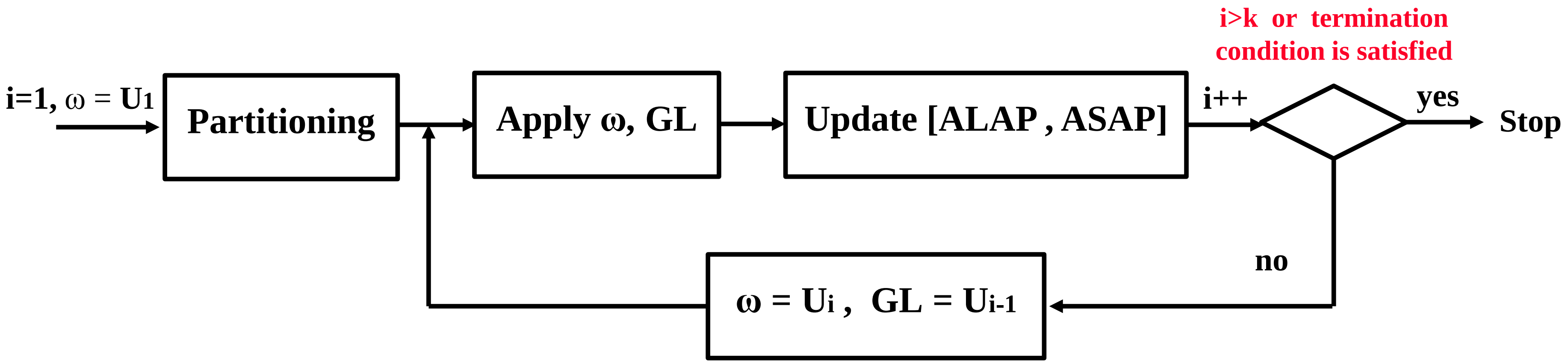


- The speculation of the j_{th} sub-task (j>i) can be $U_j'=\max(globalLow, \omega'- |i-j|)$.



Hybrid Approach

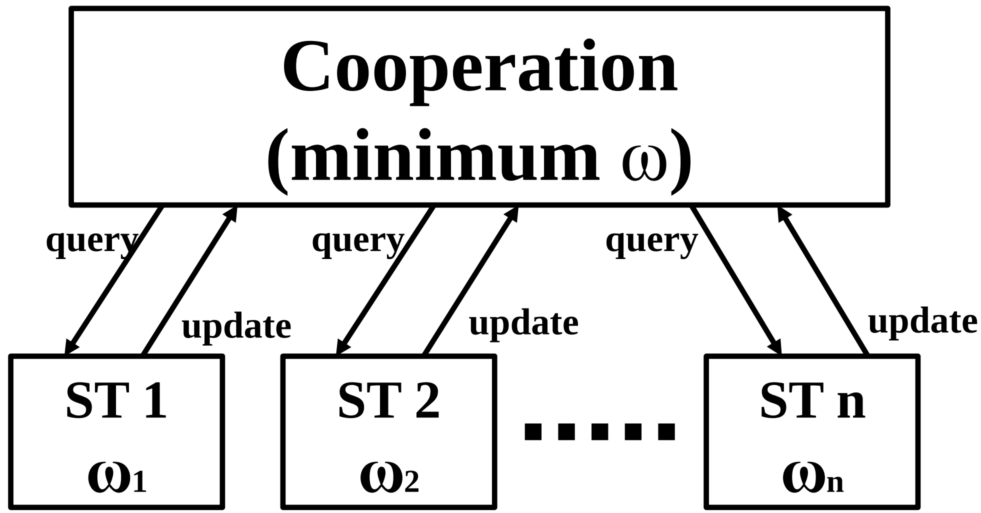
- **Search space partitioning and static upper-bound speculation approaches can be combined to further reduce the searching time.**
- **Assume that $GL=U_1<U_2<...<U_k=\omega$ are *k* upper-bound speculations. The hybrid approach has *k* iterations with increasing upper-bound sizes.**



Termination condition: find a schedule in the i_{th} iteration ($i \leq k$) and 1) ω=globalLow; or 2) all the sub-tasks in the iteration finish.

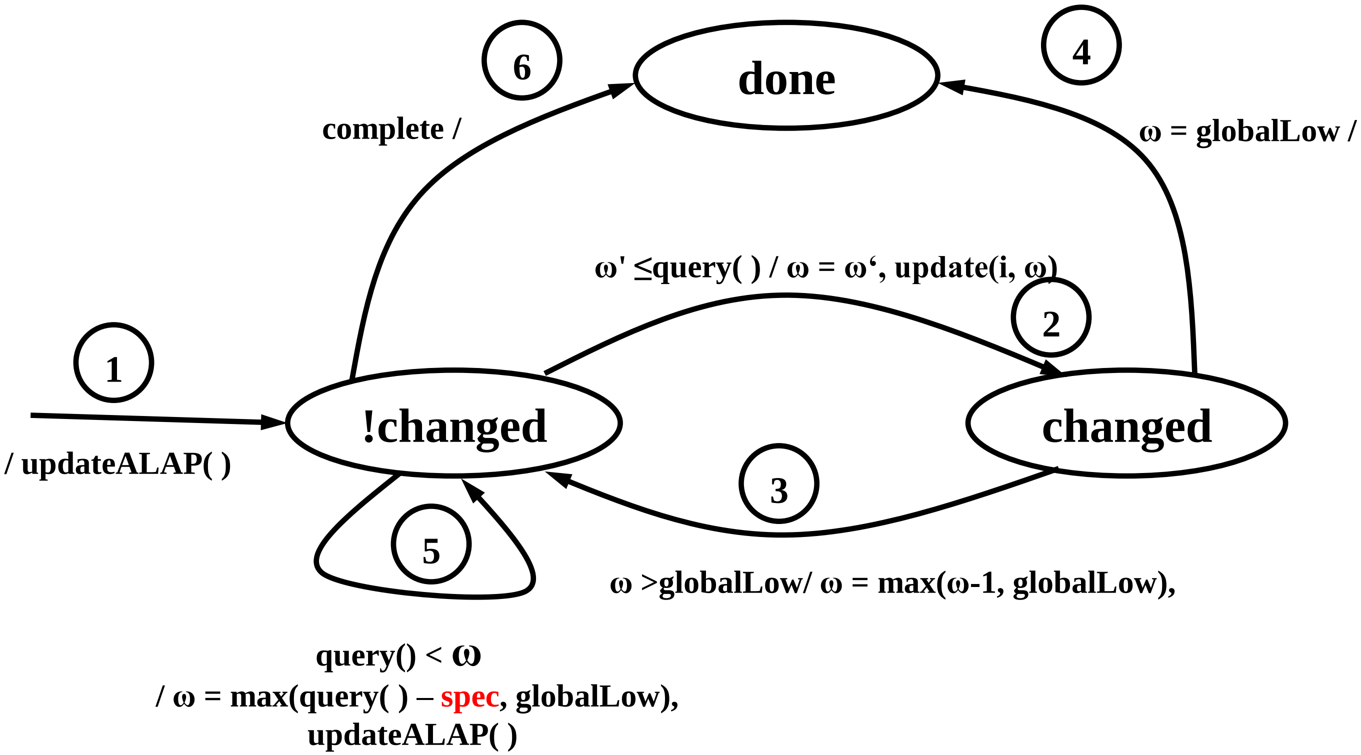
Minimum ω Synchronization

- During RCS, the search progress information (i.e., ω) of each sub-task can be different.
- If one sub-task finds a new shorter schedule (i.e., shorter ω) and such information can be propagate to other sub-tasks, the search space can be reduced drastically.



Cooperative Sub-task Implementation

- **Each sub-task is modeled using an *EFSM*.**
- **Three states: *changed* means find a better schedule with length ω'; *!changed* indicates no new better schedule since last update of ω; *done* denotes the termindation.**



Outline

- Introduction
- RCS using Branch-and-Bound Approaches
  - Graph–based Notations
  - BULB Approach
- Our Parallel Pruning Approach
  - Search Task Decomposition
  - Parallel Search Task Cooperation
- Experiments
- Conclusion

Benchmarks & Settings

- Using benchmarks from *MediaBench*.
- BULB & our approach are implemented using C++ and OpenMP.
- Experiments were conducted on a Linux server with 96 Intel Xeon 2.4GHz cores and 1T RAM.
- Setting of functional units:

Functional Unit	Operation class	Delay (unit)	Power (unit)	Energy (unit)	Area (unit)
ADD/SUB	+/-	1	**10**	10	**10**
MUL/DIV	*/	2	**20**	40	**40**
MEM	LD/STR	1	**15**	15	**20**
Shift	<</>>	1	**10**	10	**5**
Others	…	1	**10**	10	**10**

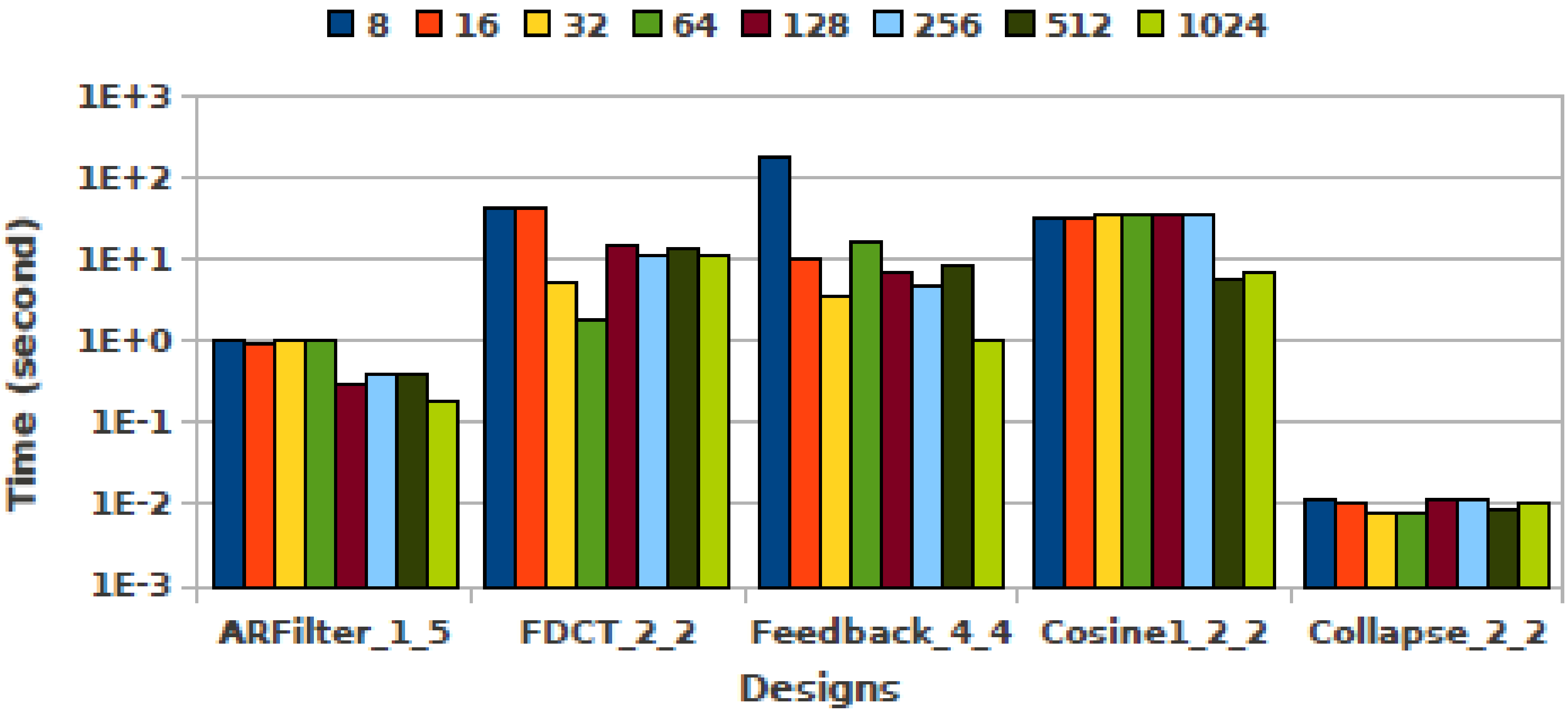
Results under Functional Constraints

Benchmark name	# of a, m	CP (sec.)	BULB (sec.)	Spec. (sec.) ssp.+dsp.	Partitioning (sec.) w/o dspec.	Partitioning (sec.) w/ dsp.	Hybrid part.+sp.	Max speedup
ARFilter	1, 3	NA	0.31	0.02	0.40	0.39	0.39	15.50
	1, 4	NA	0.78	0.06	1.00	0.97	1.01	13.00
	1, 5	NA	0.77	0.07	0.98	0.98	1.03	11.00
	2, 3	1.93	0.01	0.01	0.04	0.03	<0.01	>1.00
FDCT	1, 2	NA	36.91	43.89	<0.01	<0.01	<0.01	> 3691.00
	2, 2	NA	201.59	58.31	9.90	6.95	13.99	29.00
	2, 3	NA	19.80	6.51	7.24	3.19	2.85	6.95
	2, 4	NA	4.07	5.06	2.69	2.22	0.87	4.68
	2, 5	NA	0.92	0.99	1.10	1.07	0.04	23.00
	3, 4	NA	0.55	0.50	0.78	0.77	0.67	1.10
	4, 4	NA	0.12	0.12	0.19	0.18	0.23	1.00
Feedback	4, 4	NA	154.18	176.43	2.92	2.88	3.82	53.53
	4, 5	NA	NA	NA	3.14	3.08	4.50	3246.75
	5, 5	NA	4.87	5.50	0.35	0.35	1.51	13.92
Cosine 1	1, 2	NA	107.43	137.36	<0.01	< 0.01	< 0.01	>1.00e4
	2, 2	NA	622.83	41.02	781.34	66.74	34.54	18.03
	3, 3	NA	0.01	< 0.01	0.05	0.04	0.04	> 1.00
Collapse	2, 1	NA	NA	NA	0.04	0.03	0.02	> 5.00e5
	2, 2	NA	NA	NA	< 0.01	0.02	< 0.01	> 1.00e6
	2, 3	NA	NA	NA	<0.01	<0.01	< 0.01	> 1.00e6
	2, 4	NA	NA	NA	<0.01	< 0.01	< 0.01	> 1.00e6

RCS efforts are significantly improved with 8 cores:

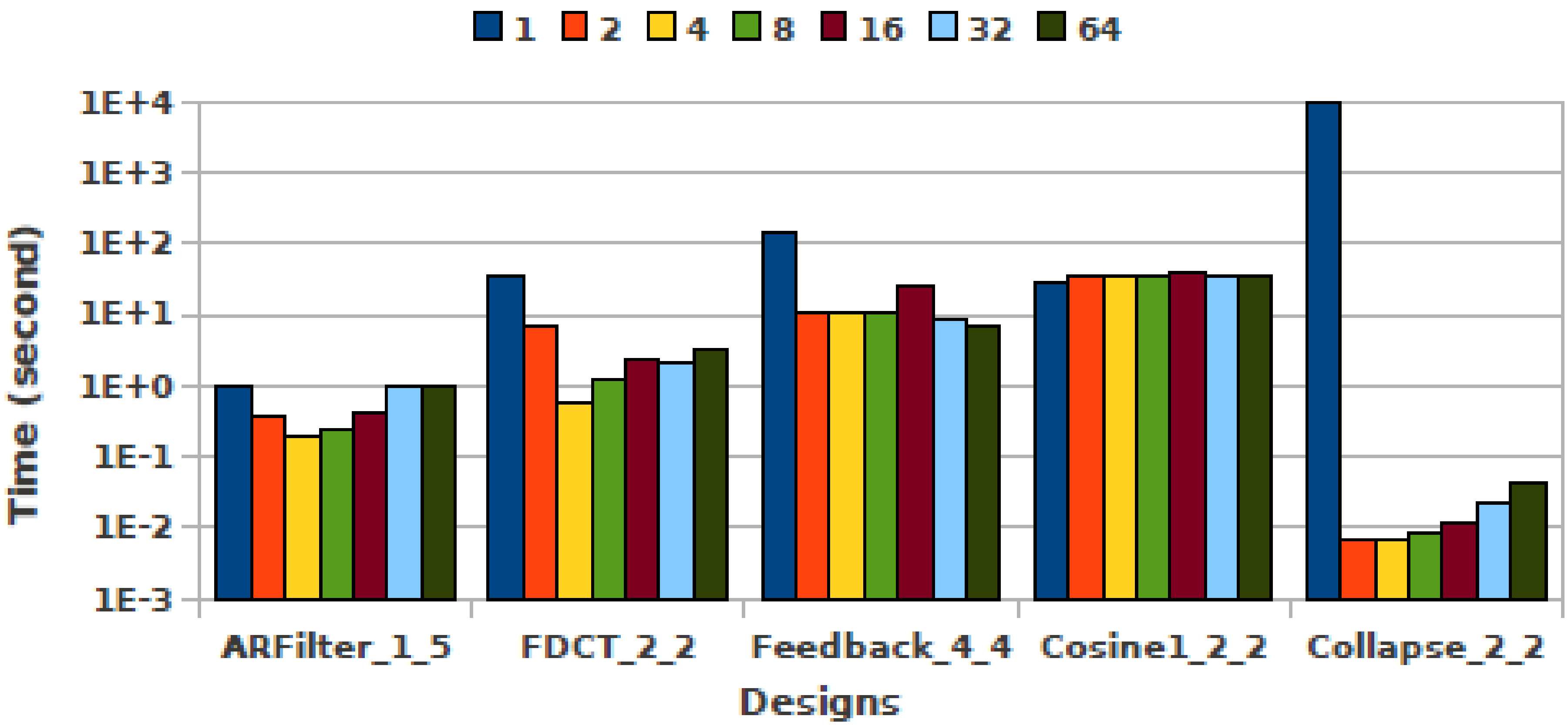
- Our parallel approaches outperform both ILP and BULB approaches
- Hybrid approach can achieve the best overall performance

Using Different Number of Partitions



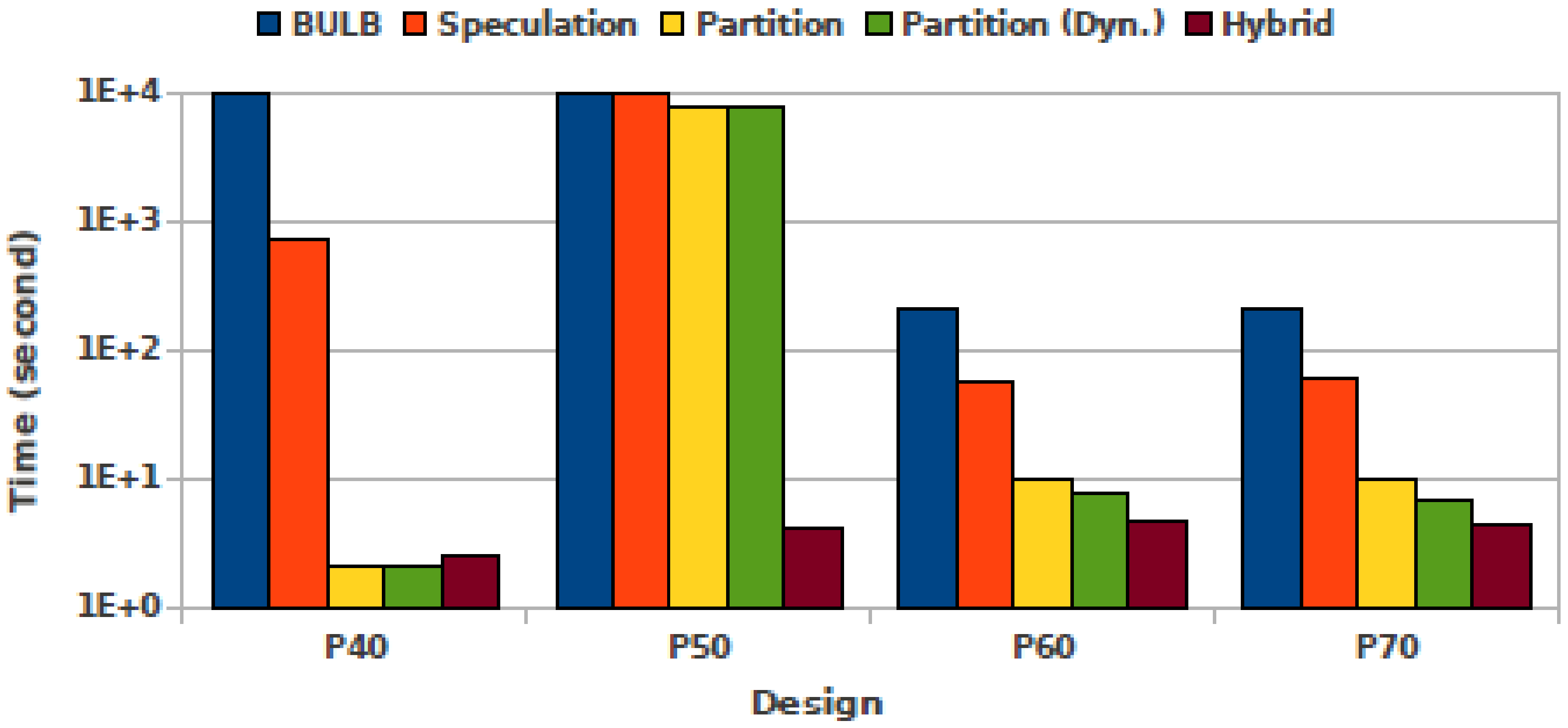
- Significant improvement using hybrid approach with 8 cores.
- When each core is assigned with ≥8 partitions, the performance will not change drastically.

Using Different Number of Cores



- The search space is divided into 128 parts.
- When the number of cores is larger than 4, increasing the core number will not reduce the search time significantly.

Scheduling Using Area of 100 Units



- FDCT design with different power and area constraints
-The hybrid approach can achieve a speedup of several orders of magnitude.

Conclusions

- **RCS is a major bottleneck in HLS**
  - Branch-and-bound approaches are promising for optimal resource-constrained scheduling
- **Proposed various parallel pruning heuristic**
  - Search space partitioning approach
  - Static /dynamic upper bound speculation approaches
  - Parallel sub-task cooperation framework
- **Successfully applied on various benchmark with different resource constraints**
  - Significant reduction in overall RCS efforts

Thank you !