Algorithms for Scheduling Task-based Applications onto Heterogeneous Many-core Architectures

Michel A. Kinsy

Department of Computer and Information Science
University of Oregon

Computer Architecture and Embedded Systems Laboratory

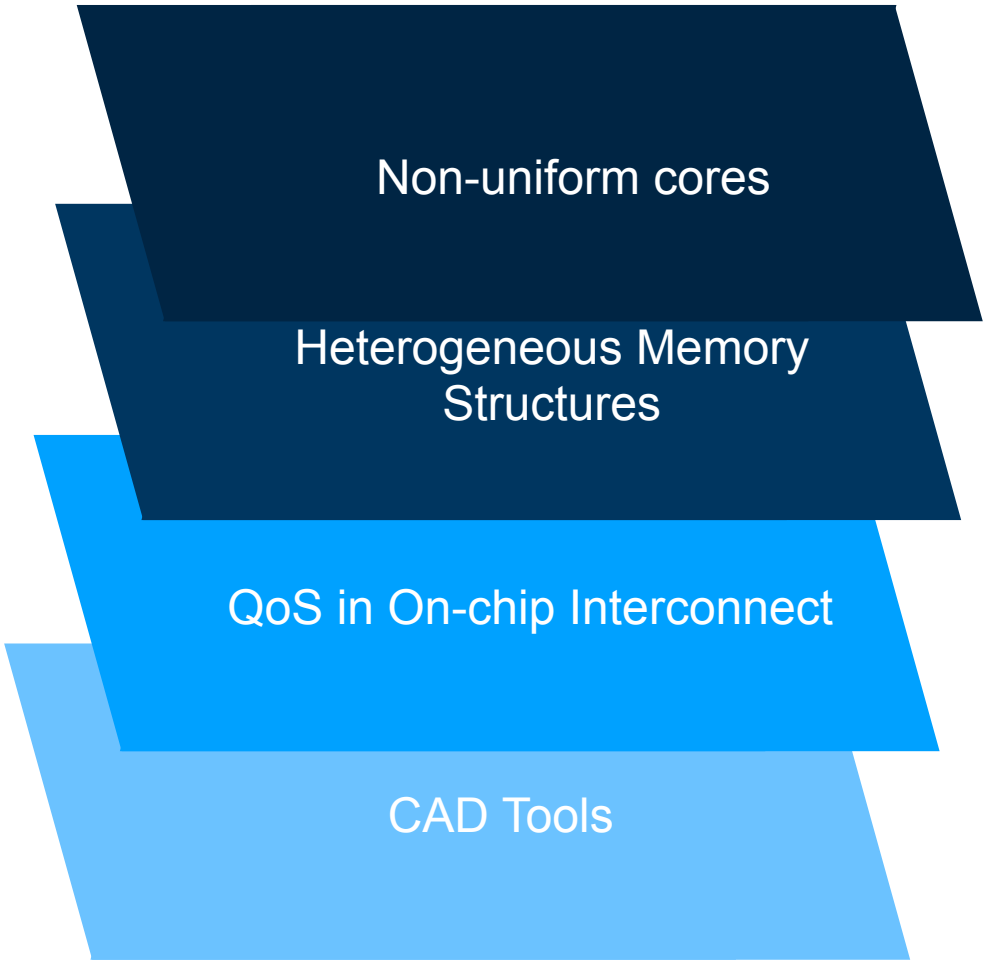
Srinivas Devadas

Department of Electrical Engineering and Computer Science
Massachusetts Institute of Technology

CSAIL

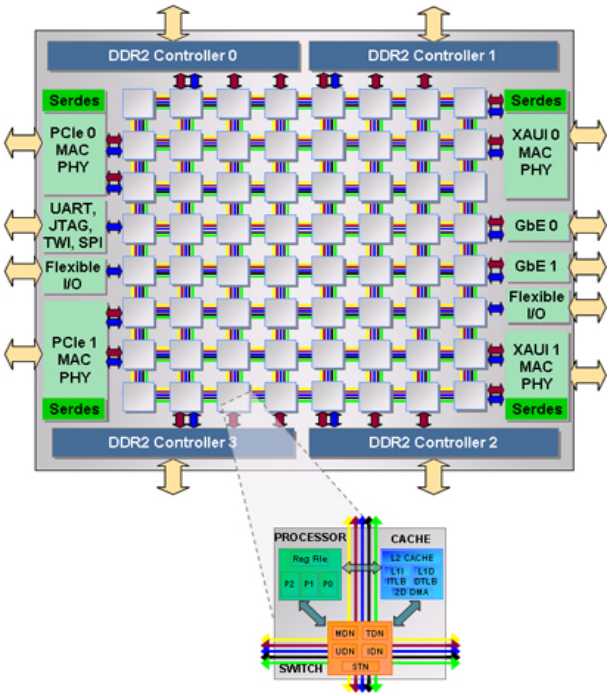
Heterogeneous Architectures Stack

- **Routing algorithms for insuring higher quality of service in on-chip communications**
- **Set of fast schedule algorithms for task-based applications onto heterogeneous architectures**

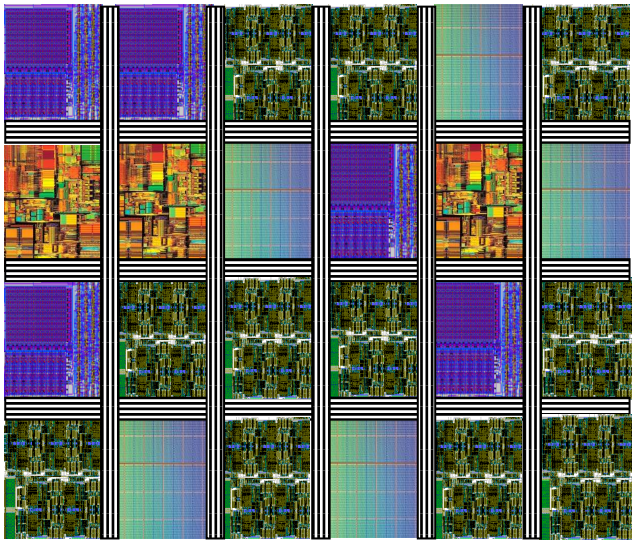


Current System Trends

- The convergence of application specific processors and multicore systems
- New phase where we see two distinctive computing architectures emerging



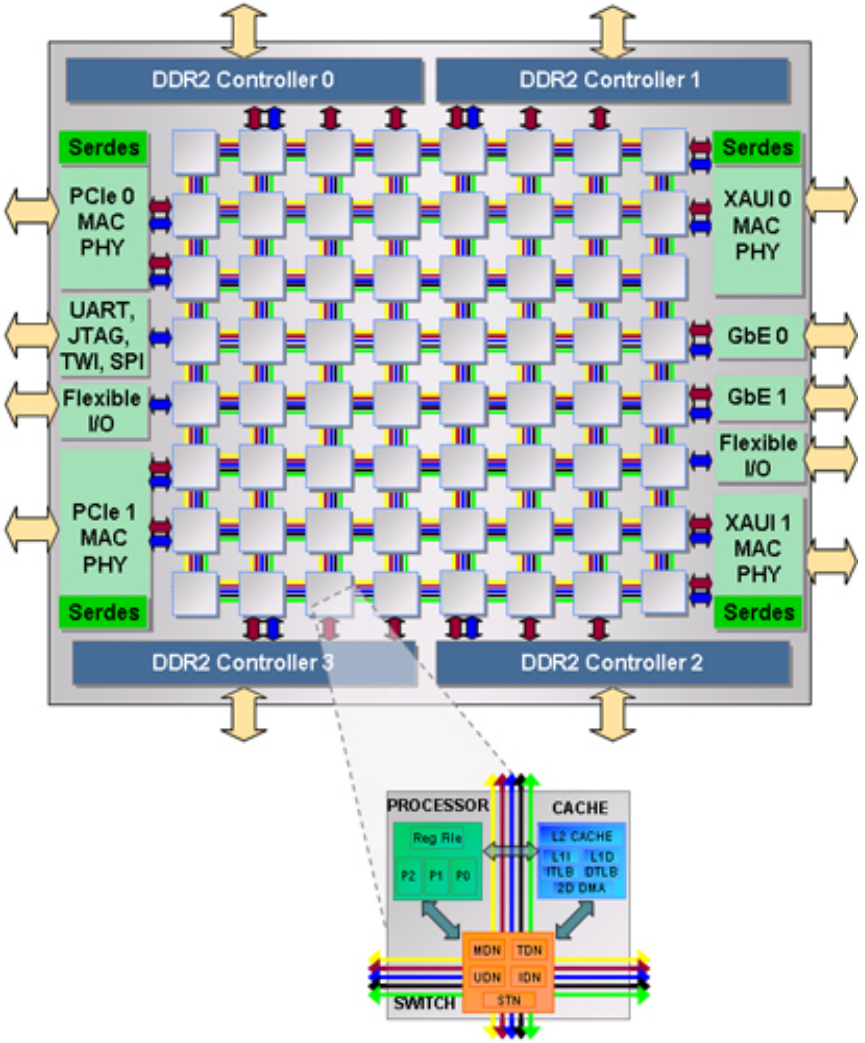
Massive parallel homogenous cores



Heterogeneous many-core architectures

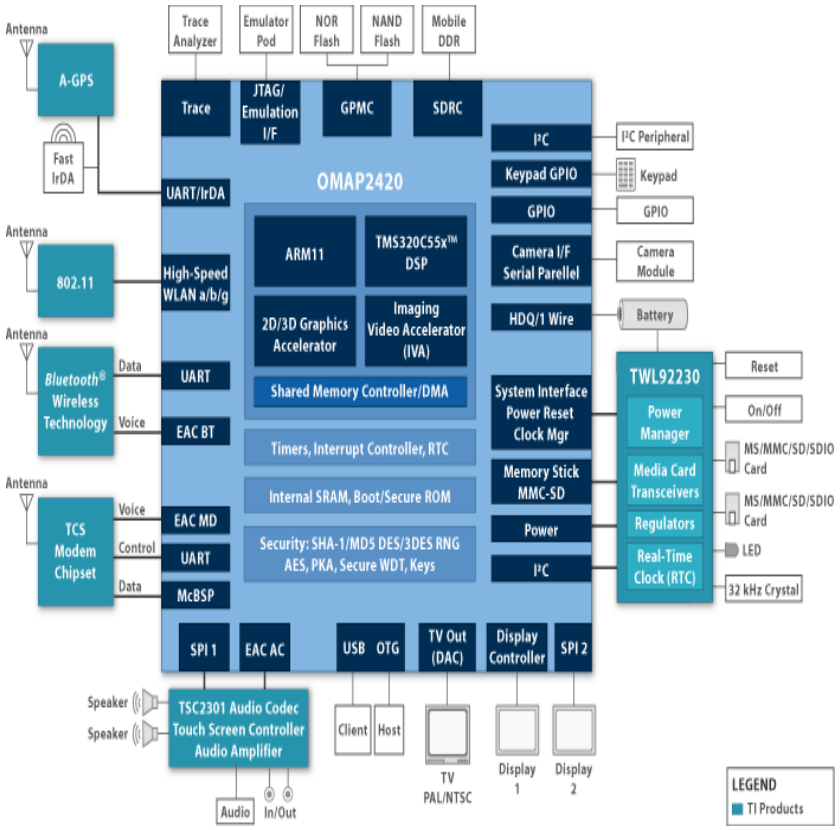
Homogeneous Many-cores

- Tilera-like homogenous architectures: servers, cloud computing…
- Great for trivially parallel applications
- Having similar cores on the die makes the manufacturing and testing process more manageable
- Homogeneous collection of cores also keeps the software support model simple
- **Lack of specialization of hardware to different tasks**



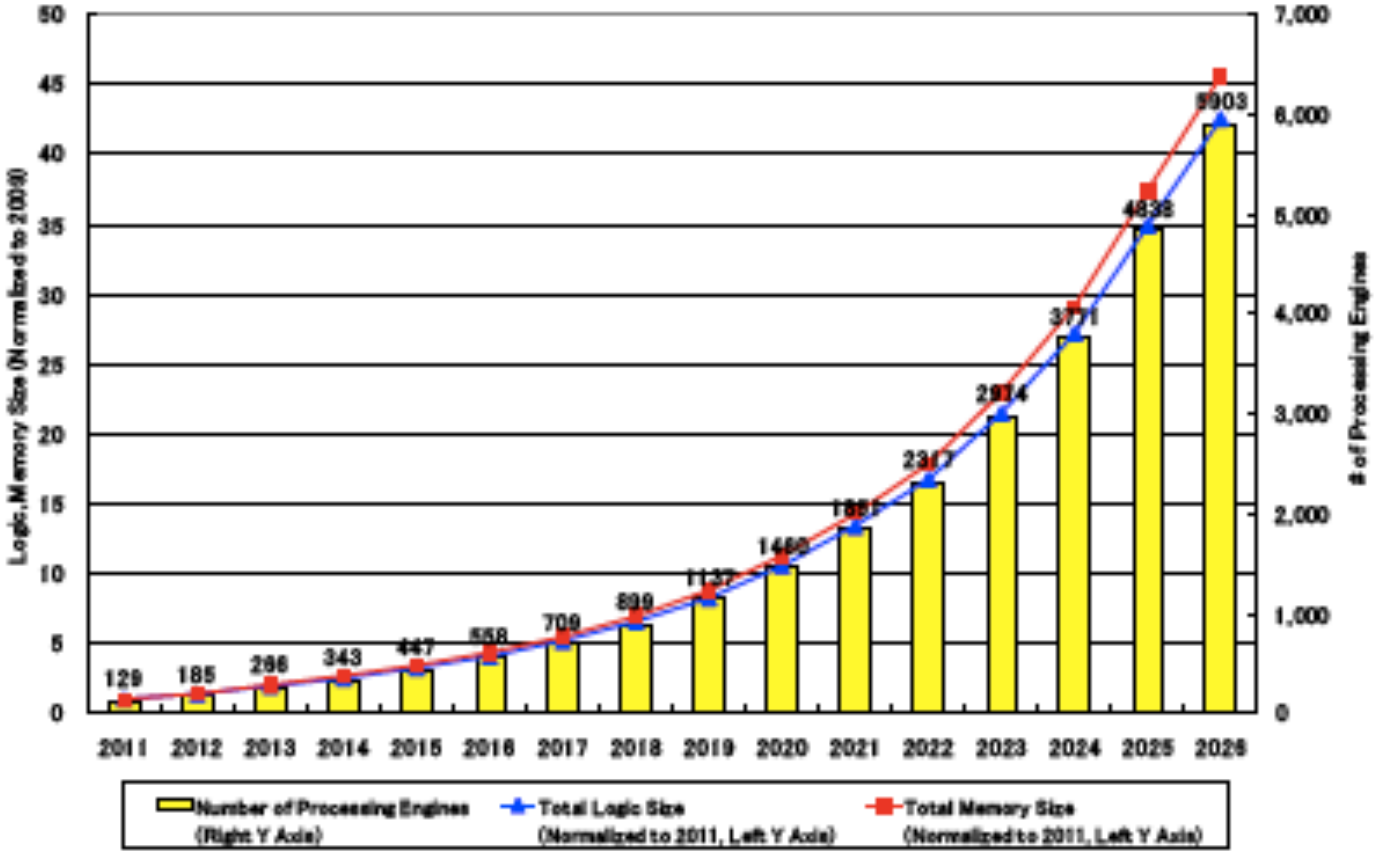
Heterogeneous Many-cores

- Integration of heterogeneous technologies
- These SoC architectures have large number of processing units
  - programmable RISC/CISC cores, memory, DSPs, and accelerator function units/ASIC
- Logic diagram of the hardware processing units in a typical smart phone today



Complex SoC Architectures

- **SoC architectures need more research and standardization to encourage high degree of reusability**



SoC design complexity trends [International Technology Roadmap for Semiconductors 2011 Report]

Advantages of Heterogeneity

- Applications are naturally heterogeneous
- Performance and power can be better optimized
- Specialization of compute units to different tasks promises greater energy/area efficiency
- Kumar et al.* present a heterogeneous architecture which outperforms a comparable-area homogeneous architecture by up to 63%
- And with a good task-to-core assignment, they are able to achieve an additional 31% speedup over a naive scheduling policy
- But scheduling on heterogeneous resources is NP-complete in general cases, as well as in several restricted cases

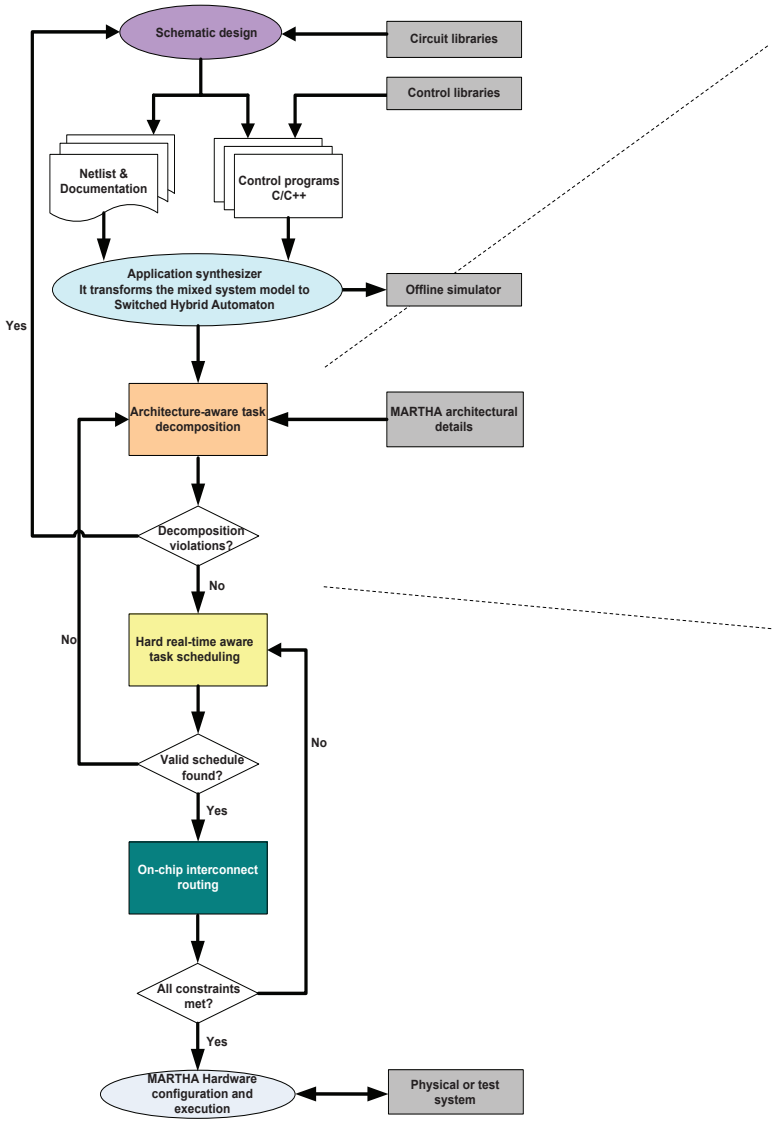
* Kumar, Rakesh and Tullsen, Dean M. and Ranganathan, Parthasarathy and Jouppi, Norman P. and Farkas, Keith I.: "Single-ISA Heterogeneous Multi-Core Architectures for Multithreaded Workload Performance", ISCA '04.

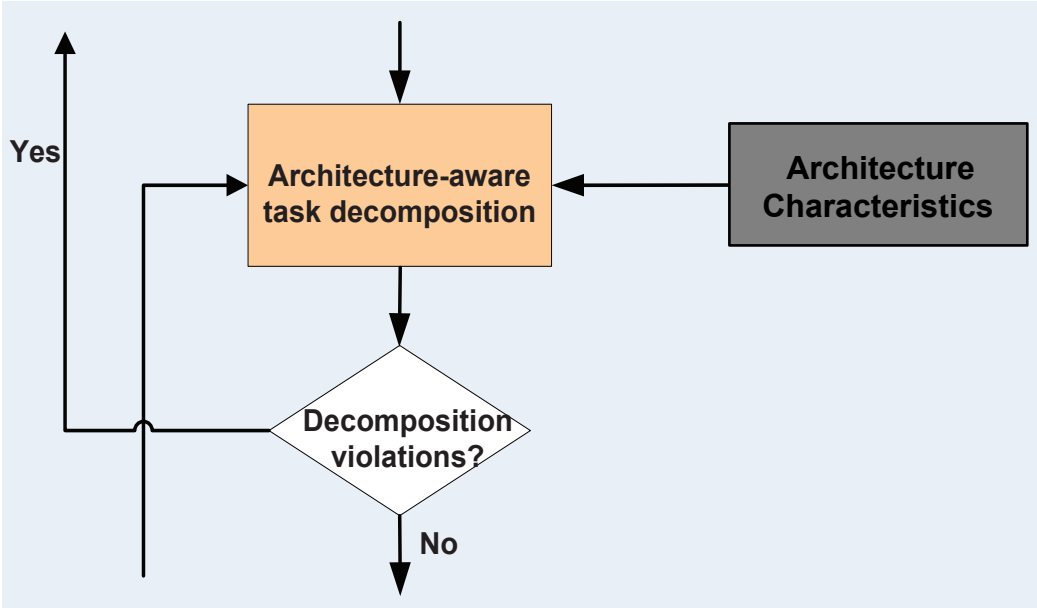
Design Challenges

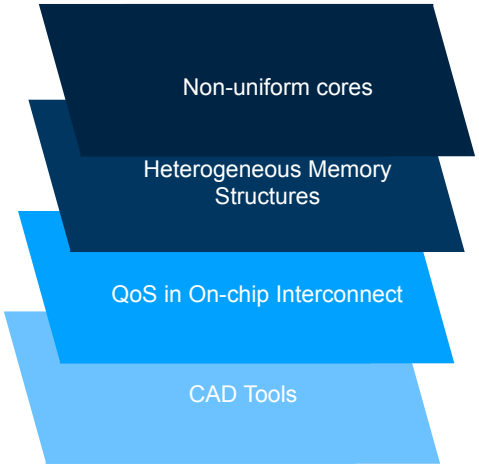
- Design of heterogeneous many-core systems shares many of the same challenges in general-purpose homogeneous architectures, but there are few added design constraints
- SoC architectures are deployed in many computation environments that require concurrent execution of several tasks with different, sometimes opposite, performance goals
- Integration of different services on the same computing platform requires rethinking of the cores, the memory hierarchy, and the interconnect network
- On these platforms, we need to think not only in terms of tasks and task parallelism, but also services and service guarantees

- Task Scheduling for efficient and effective power and performance management

CAD Tools: Task Scheduling

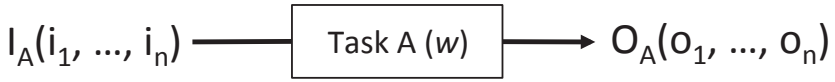






Task Decomposition

- **An application is defined as a composition of computation tasks to provide one global compute service**
- **A task is a primitive computation object**



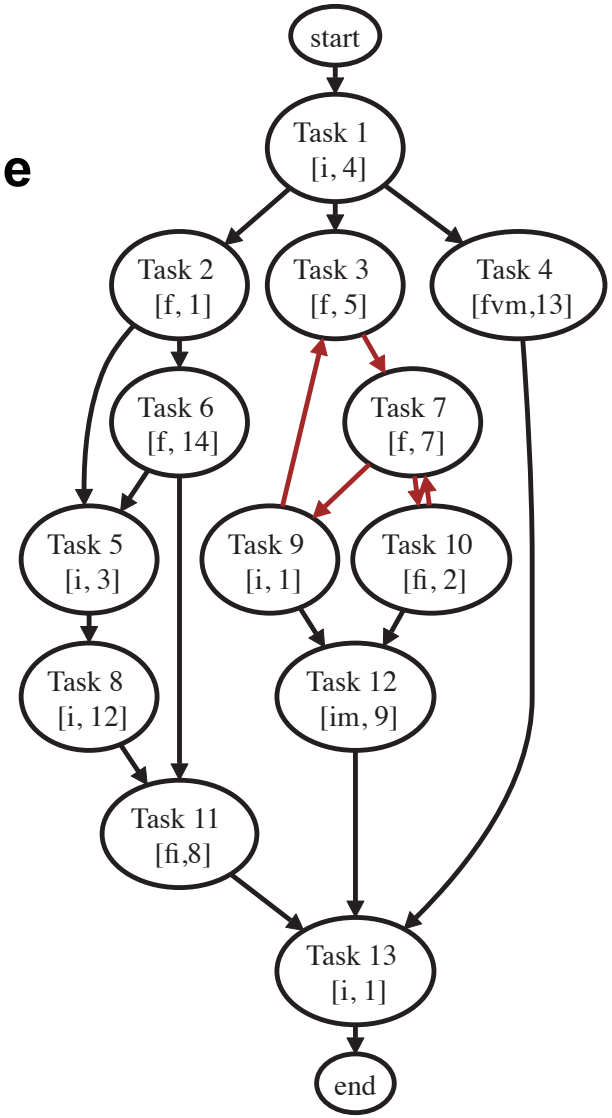
- **There are three elementary tasks:**
  - **feedback task** $I(i_1, i_2, ..., i_n)_A^t \cap O(o_1, o_2, ..., o_m)_A^{t-1} \neq \phi$
  - **sequential task**
  - **parallel task**

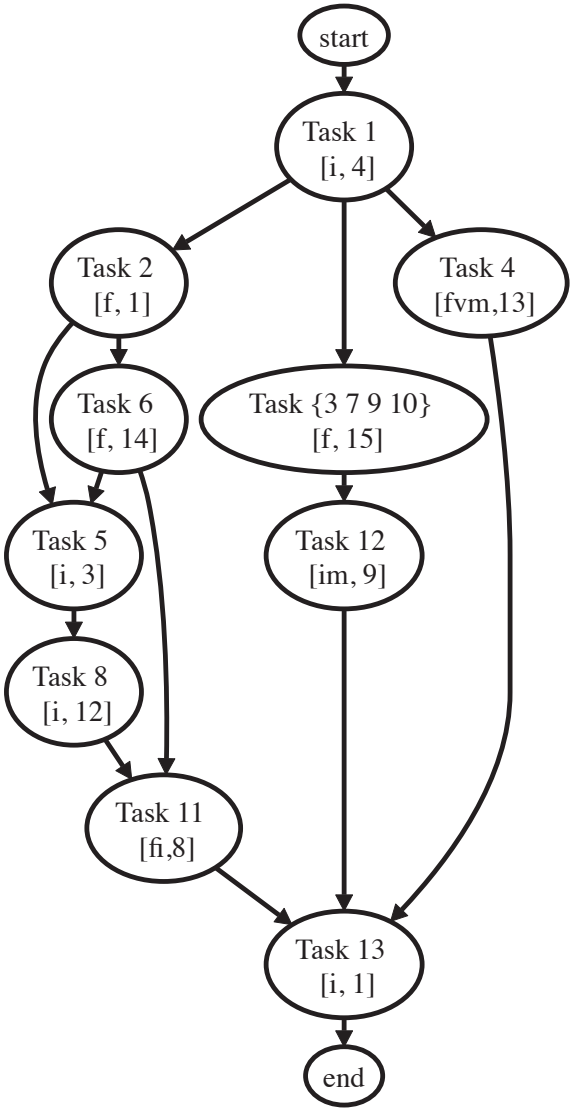
- **Task fusion: use to force a set of tasks must be assigned to the same processing** $O(o_1, o_2, ..., o_m)_A^t = I(i_1, i_2, ..., i_n)_B^t$

$C \; O(o_1, o_2, ..., o_m)_A^t \cap I(i_1, i_2, ..., i_n)_C^t = \phi \quad O(o_1, o_2, ..., o_m)_A^{t-1} \cap I(i_1, i_2, ..., i_n)_C^t = \phi$

Task Decomposition: Task Fusion

- **Using task fusion the task decomposition step is able to generate multiple equivalent DAGs**





Taxonomy

- Processor allocation: on which processor a task should execute.
- Task priority: when, and in what order with respect to other tasks, should each task execute.
- Fixed task priority: Each task has a single priority that remains the same through all phases of the application
- Preemptive: tasks can be preempted by a higher priority task at any time.
- Task-level migration: threads of a task may execute on different processors
  - However each thread can only execute on a single processor
- See paper for full list

Formulation of Task-Based Scheduling

- $A = (T_1, T_2, \ldots, T_k)$ where task $T_i = (w_i, d_i, r_i, c_i)$
  - w_i (working set of task T_i)
  - d_i (deadline of task T_i),
  - r_i (input rate of task T_i)
  - c_i (instruction count in task T_i)

- A processing element $p_i = (\rho_i, l_{mini}, l_{maxi})$, where $\rho_i = f(IPC, EUs, CSs)$
  - L_{min}: memory access latency on a cache hit
  - L_{max}: maximum miss latency.
  - IPC: instructions-per-cycle
  - EU: execution unit (set of execution functional units, e.g., floating-point unit)
  - CS: cache sizes

Scheduling Definitions

- A task admissible schedule (TAS) for an application A is a set of tuples that associates to each task T a set of processing elements such that the data dependencies and timing constraints between tasks are respected
- A task T_i is schedulable on a processing element p_j, denoted $T_i \triangleright p_j$, if its worst-case execution time p_j is less than or equal to its deadline
- An application is schedulable according to a TAS algorithm if all of its tasks are schedulable
- An application is said to be feasible with respect to a given heterogeneous many-core system if there exists at least one TAS solution that can schedule all possible sequences of tasks that may be generated by the application on that system without missing any deadlines or violating any inter-task dependency

Algorithms

- **Integer Linear Programming Formulation**

The objective function is:

$$minimize \quad T_k(t_f) \tag{1}$$

Subject to:

$$T_i(t_f) \leq d_j \tag{2}$$

$$T_i(t_s) \leq T_i(t_f) \tag{3}$$

$$if\ e(T_i, T_j) \in E \qquad T_i(t_f) < T_j(t_s) \tag{4}$$

$$\forall\ e(T_i, T_j) \in E, \quad T_j(t_s) - T_i(t_f) = d_{i,j} \tag{5}$$

$$\forall\ (P(T_i) = p_u\ ,\ P(T_j) = p_v), \quad w_{i,j} \times b_{p_u,p_v} \leq d_{i,j} \tag{6}$$

$$\begin{aligned} \forall\ T_i \in A, \forall p_u \in P, \\ E_{(T_i,p_u)} &= (\wp_u \times c_i) \\ &+= (w_i \times hit_{rate_{i,u}} \times l_{min_u}) \\ &+= (w_i \times (1 - hit_{rate_{i,u}}) \times l_{max_u}) \end{aligned} \tag{7}$$

$$For \quad T_i \in A,\ p_u \in P, \quad T_i(t_f) - T_i(t_s) \geq E_{(T_i,p_u)} \tag{8}$$

$$\forall\ T_i \in A,\ p_u \in P, \quad M(T_i, p_u) - (E_{(T_i,p_u)} \times b_{i,u}) = 0 \tag{9}$$

$$\forall\ T_i \in A, \quad \sum_{u=1}^{n} b_{i,u} = 1 \tag{10}$$

$$\forall\ p_u \in P, \quad \sum_{i=1}^{k} M(T_i, p_u) \leq T_k(t_f) \tag{11}$$

Algorithms

- **Integer Linear Programming Formulation**

The objective function is:

$$minimize \quad T_k(t_f) \tag{1}$$

Subject to:

$$T_i(t_f) \leq d_j \tag{2}$$

$$T_i(t_s) \leq T_i(t_f) \tag{3}$$

$$if\ e(T_i, T_j) \in E \qquad T_i(t_f) < T_j(t_s) \tag{4}$$

$$\forall\ e(T_i, T_j) \in E, \quad T_j(t_s) - T_i(t_f) = d_{i,j} \tag{5}$$

$$\forall\ (P(T_i) = p_u\ ,\ P(T_j) = p_v), \quad w_{i,j} \times b_{p_u,p_v} \leq d_{i,j} \tag{6}$$

$$\forall\ T_i \in A, \forall p_u \in P,$$

$$\begin{aligned} E_{(T_i,p_u)} &= (\wp_u \times c_i) \\ &+= (w_i \times hit_{rate_{i,u}} \times l_{min_u}) \\ &+= (w_i \times (1 - hit_{rate_{i,u}}) \times l_{max_u}) \end{aligned} \tag{7}$$

$$For \quad T_i \in A,\ p_u \in P, \quad T_i(t_f) - T_i(t_s) \geq E_{(T_i,p_u)} \tag{8}$$

$$\forall\ T_i \in A,\ p_u \in P, \quad M(T_i, p_u) - (E_{(T_i,p_u)} \times b_{i,u}) = 0 \tag{9}$$

$$\forall\ T_i \in A, \quad \sum_{u=1}^{n} b_{i,u} = 1 \tag{10}$$

$$\forall\ p_u \in P, \quad \sum_{i=1}^{k} M(T_i, p_u) \leq T_k(t_f) \tag{11}$$

Algorithms

- **Simple fast heuristics for large size problems**

Algorithm 1 Minimizes intersections across all mapping sets (H1).

Assumption: An application A composed of a number of tasks, $A = \{T_1, T_2, ..., T_k\}$ and a system with a list of processing elements $P = \{p_1, p_2, ..., p_n\}$.

Objective: Find a set of task-processor mapping $S = \{S(T_1), S(T_2), ..., S(T_k)\}$ such that: $\forall\ T_i \in A,\ S(T_i) = \{p_u, ..., p_v\}$ with $1 \leq u < v \leq n$ while minimizing $\forall (i,j)\ S(T_i) \cap S(T_j)$.

Begin
$\forall\ T_i \in A,\quad S(T_i) = \phi$
for $i = 1;\ i \leq k;\ i++$: **do**
  for $j = 1;\ j \leq n;\ j++$: **do**
    if $(T_i \triangleright p_j)$ **then**
      $S(T_i) = S(T_i) \cup \{p_j\}$
    end if
  end for
end for

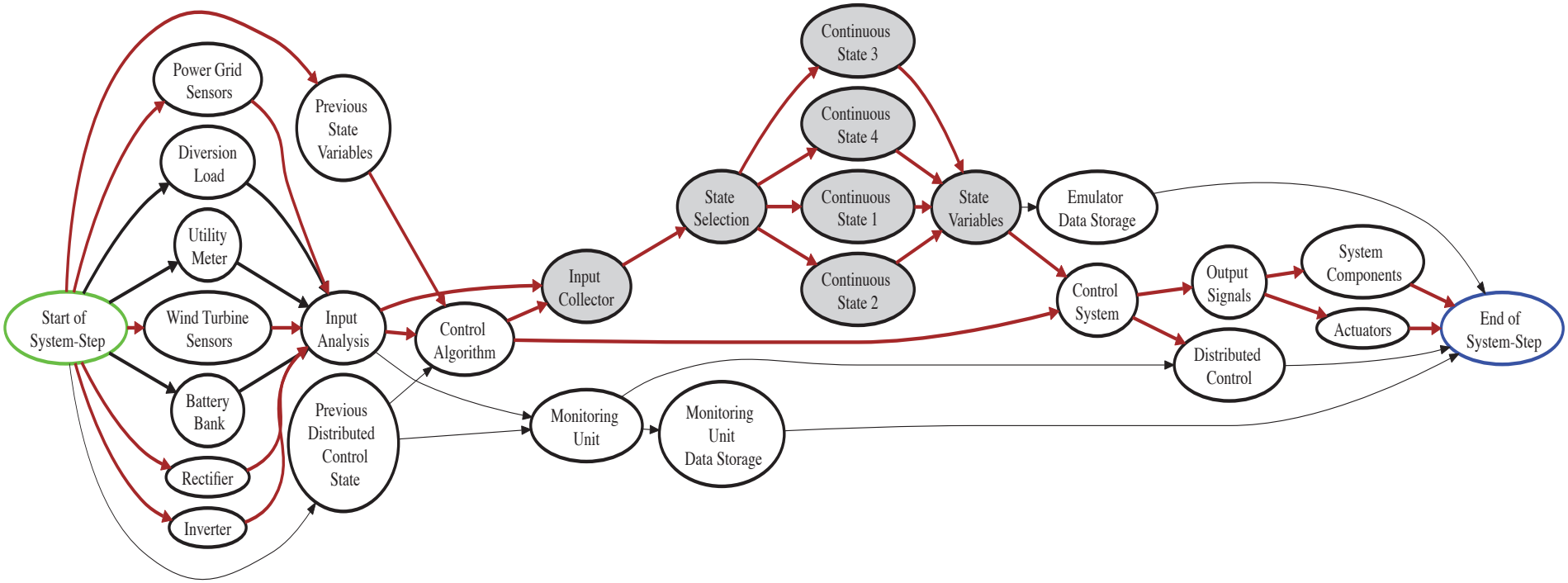
while $(\forall\ T_i \in A,\ |S(T_i)| > 1$ and $\forall\ (i,j)\ S(T_i) \cap S(T_j) \neq \phi)$ **do**
  if $(\exists\ (T_i, T_j) \mid S(T_i) \cap S(T_j) \neq \phi)$ **then**
    if $((|S(T_i)| > 1) \wedge (|S(T_j)| > 1))$ **then**

$$S(T_i) = \begin{cases} S(T_i) - \{S(T_{min}) \cap S(T_i)\}\ where \\ S(T_i) \subsetneq \{S(T_{min}) \cap S(T_i)\} \\ \\ \{p_e\}\ for\ any\ p_e \in S(T_i)\ otherwise \end{cases}$$

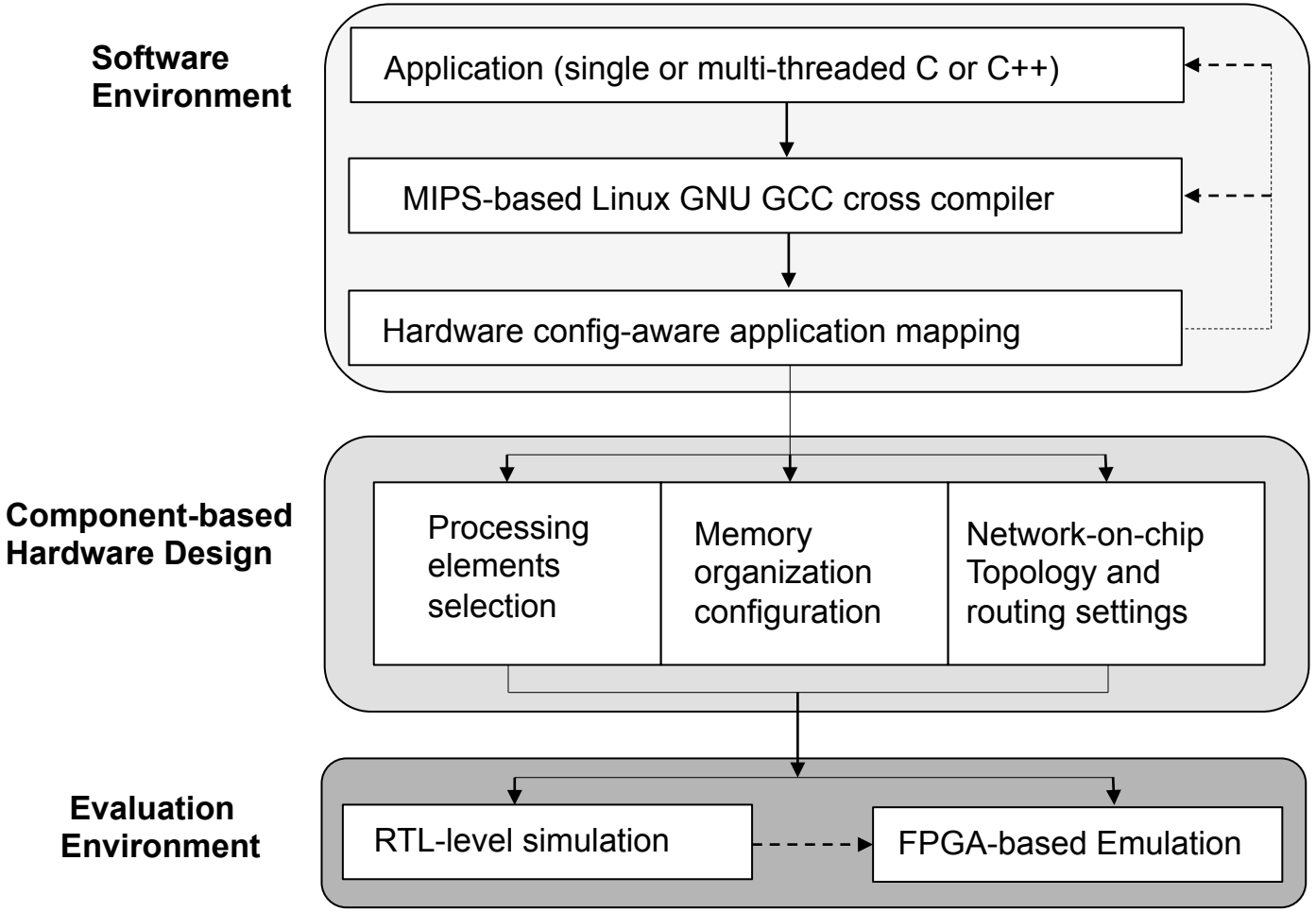
    end if
  end if
end while
End

Illustration Through Application

- **Wind turbine application acyclic task graph**



Heracles* Design Environment



http://projects.csail.mit.edu/heracles

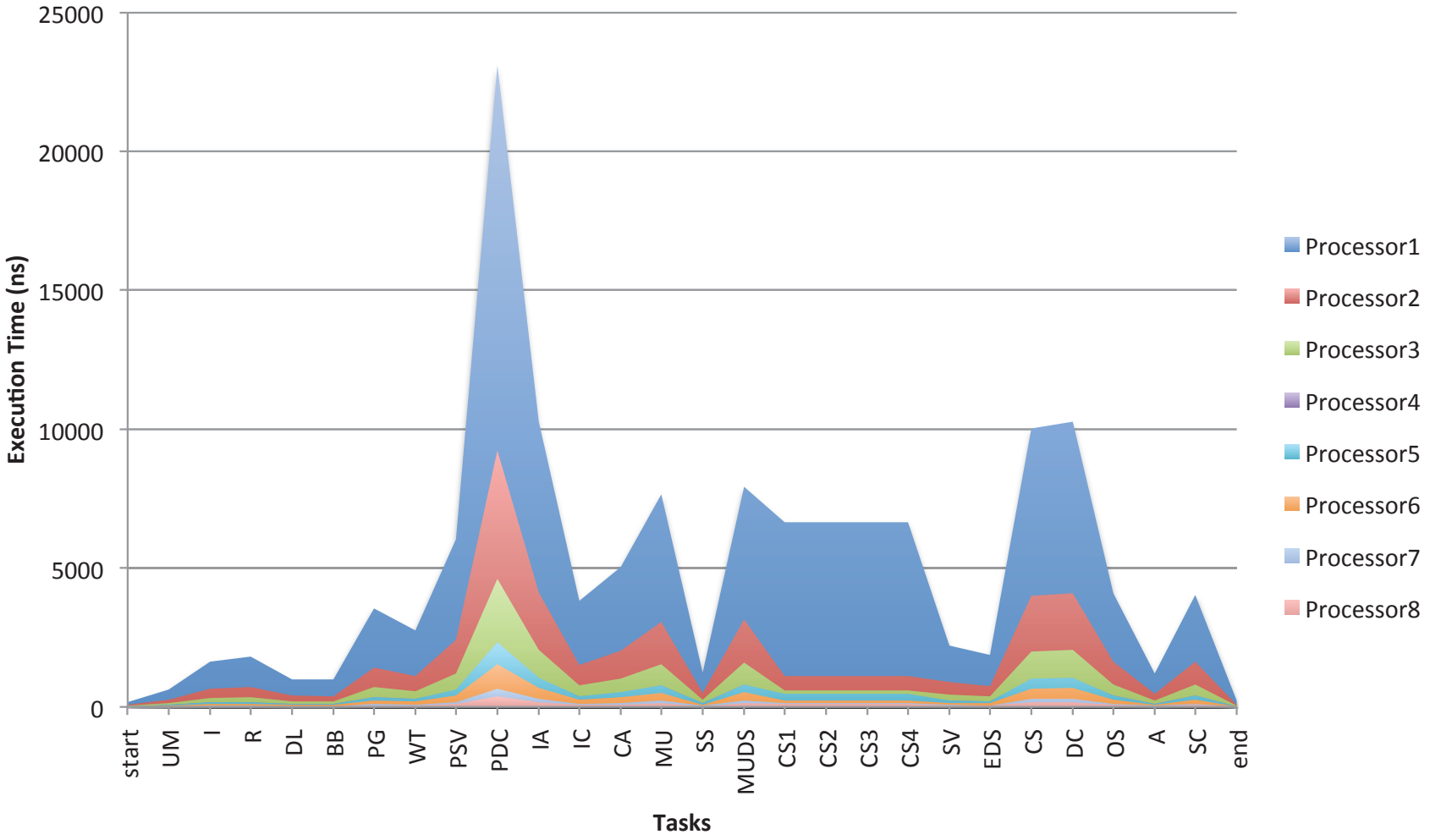
[*] M. Kinsy, M. Pellauer, and S. Devadas: "Heracles: A Tool for Fast RTL-Based Design Space Exploration of Multicore Processors." In Proceedings of the 21st International Symposium on **Field Programmable Gate Arrays (FPGA)**, February 2013

Architecture Components

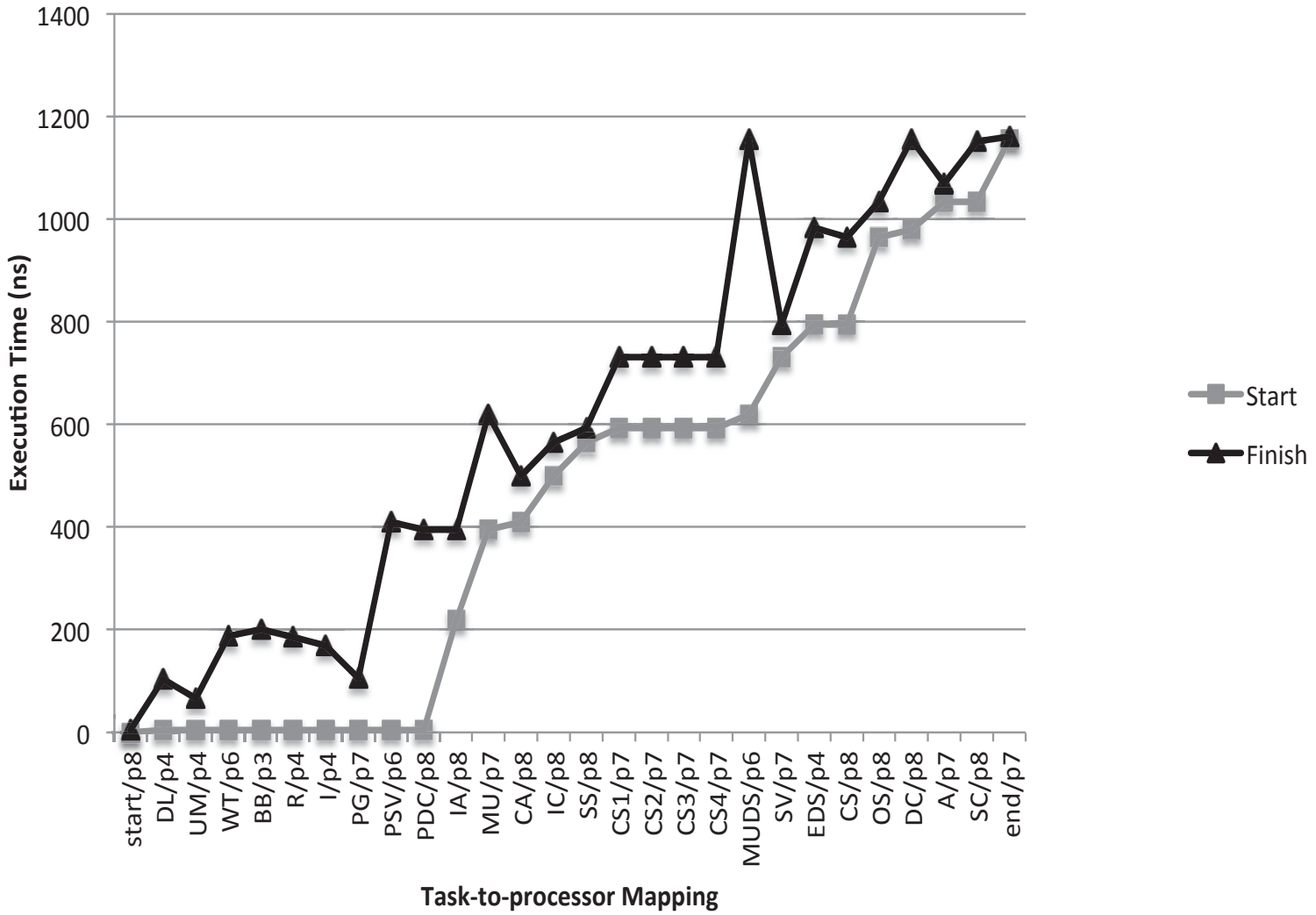
- **Using the Heracles multicore design platform we construct eight different processor core configurations**

1. **16-bit microprocessor (Processor1)**
2. **single-cycle MIPS core (Processor2)**
3. **7-stage single-threaded MIPS core (Processor3)**
4. **7-stage 2-way threaded MIPS core (Processor4)**
5. **Single-lane vector machine (Processor5)**
6. **2-lane vector machine (Processor6)**
7. **4-lane vector machine (Processor7)**
8. **8-lane vector machine (Processor8)**

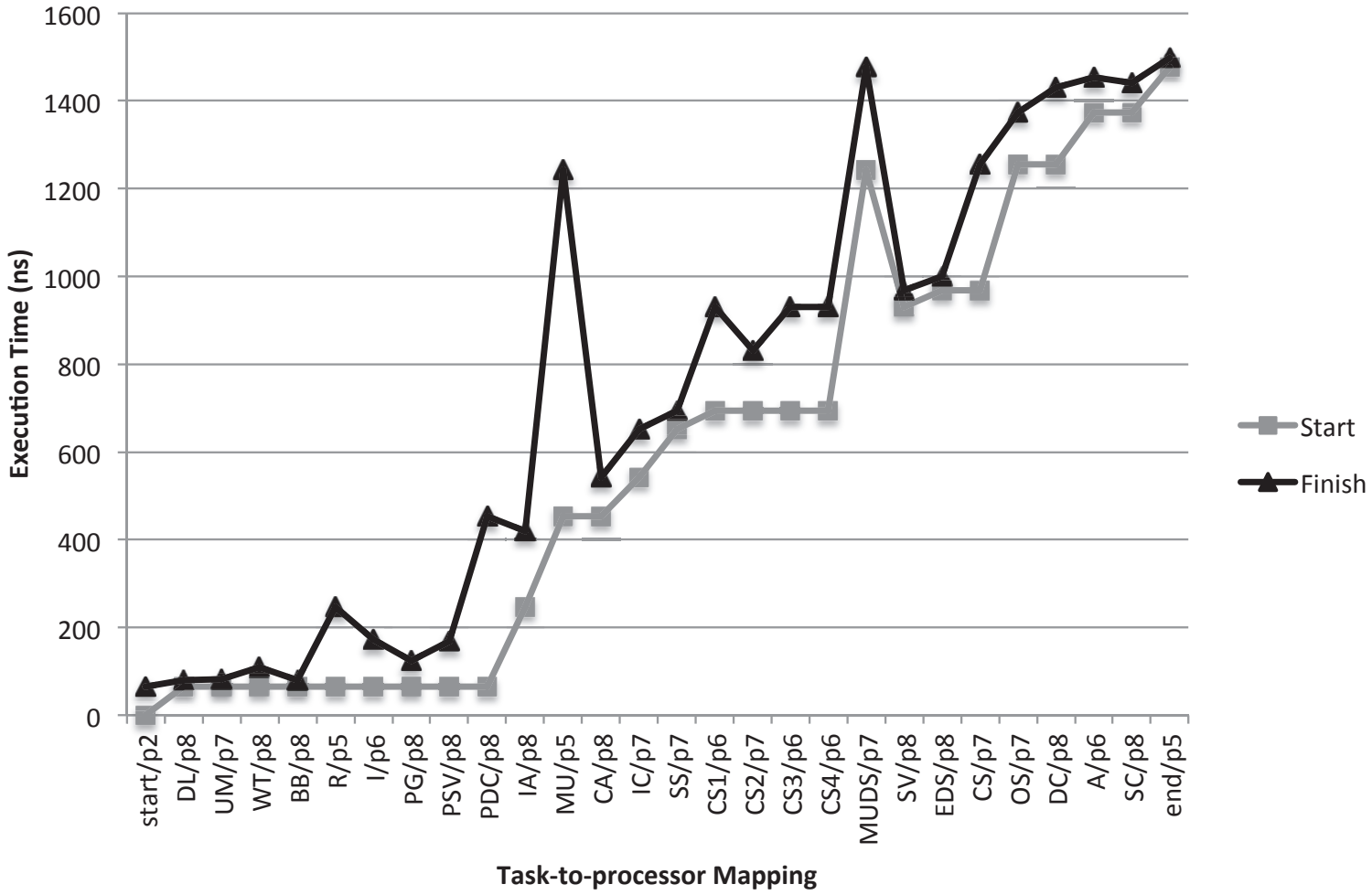
Wind Turbine Execution per Type



ILP-Based Scheduling



Heuristic 2-Based Scheduling



Q & A Section

Thank You !

More Information at
http://caes.cs.uoregon.edu/