A CnC-driven Implementation of Medical Imaging Algorithms on Heterogeneous Processors

Yi Zou[*], Zoran Budimlić[+], Alina Sbîrlea[+], Sağnak Taşırlar[+], Vivek Sarkar[+]

[*]University of California at Los Angeles

[+]Rice University

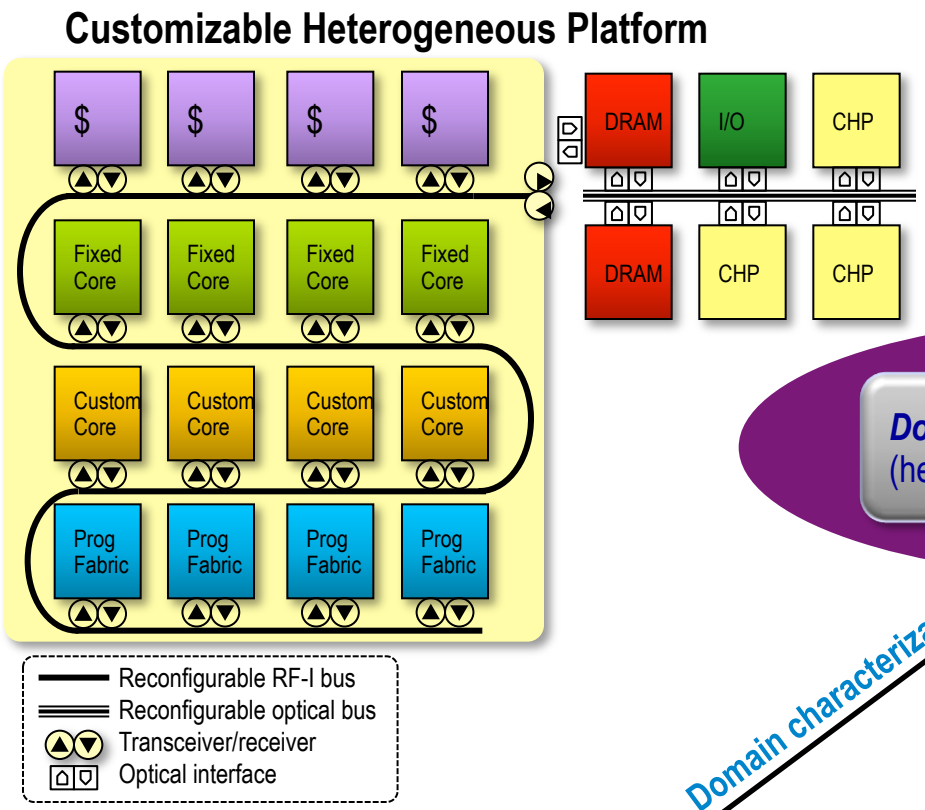
RICE

Outline

- **Domain-Specific Computation**
- Medical Imaging Pipeline
- CnC Model of the Medical Imaging Pipeline
- Locality and Heterogeneity: Hierarchical Place Trees
- Experimental Results and Conclusions

RICE

Domain-Specific Modeling



Modeling

Domain-specific-modeling
(healthcare applications)

Mapping

Domain characterization

Application modeling

CHP creation
Customizable computing engines
Customizable interconnects

Design once (configure)

Architecture modeling

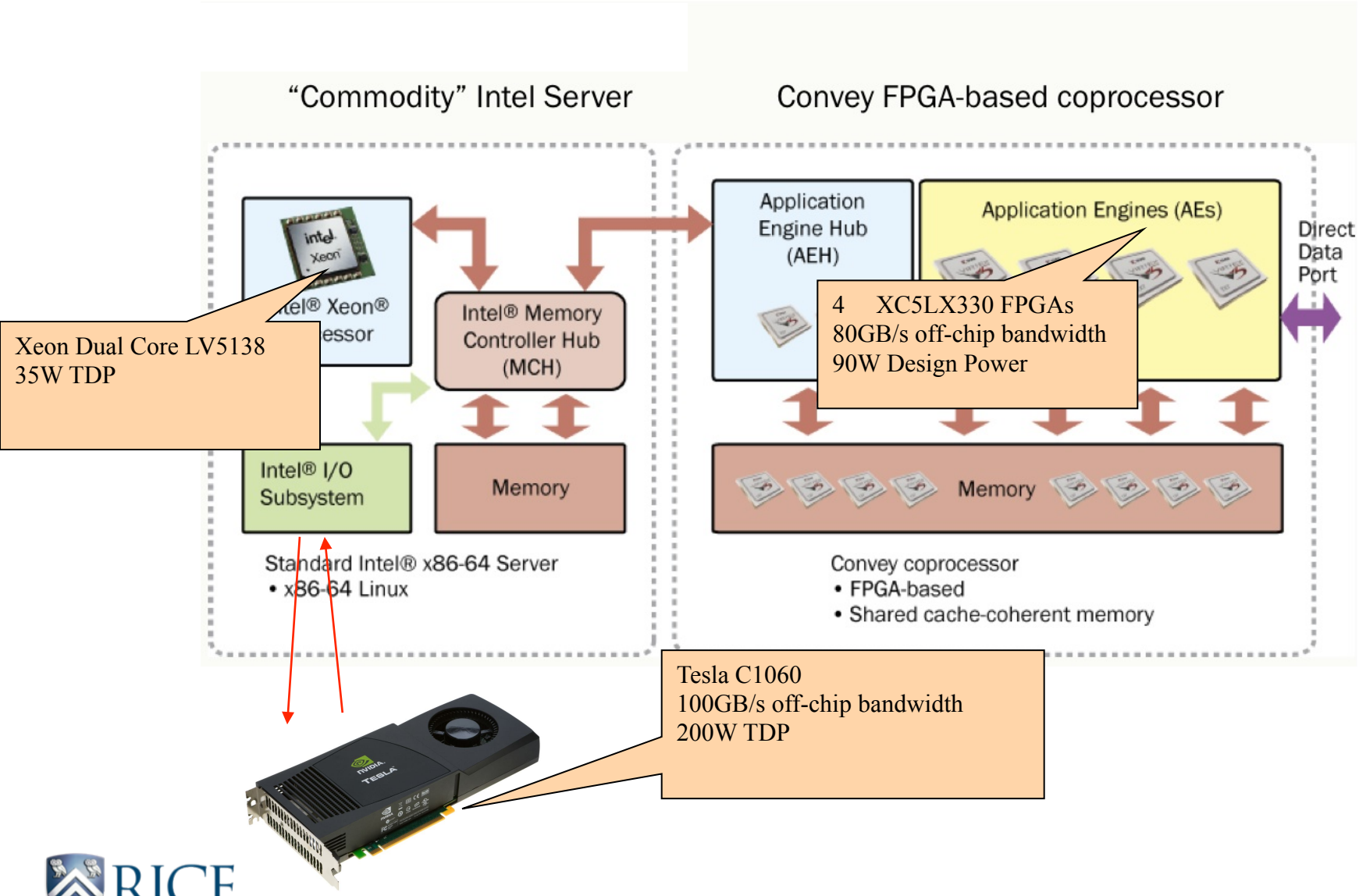
Customization setting

CHP mapping
Source-to-source CHP mapper
Reconfiguring & optimizing backend
Adaptive runtime

Invoke many times (customize)

RICE

Heterogeneous Server Testbed HC-1 Architecture



RICE

Outline

- Domain-Specific Computation
- **Medical Imaging Pipeline**
- CnC Model of the Medical Imaging Pipeline
- Locality and Heterogeneity: Hierarchical Place Trees
- Experimental Results and Conclusions

RICE

Case Study: Medical Imaging Pipeline

- **Medical image processing pipeline**
  - Covering all imaging tasks: reconstruction, denoising/deblurring, registration, segmentation, and analysis
  - Each task can involve the use of different algorithms, dependent on the data and disease domain
  - Initially targeting automated volumetric tumor assessment for cancer
- **Base sequential pipeline**
  - C/C++ with a common data API to wrap each algorithm (handles image and parameter passing; result output)
  - Java Native Interfaces (JNI) is used to execute the pipeline from an image viewing application

RICE

Pipeline Algorithms

	Algorithm	Language(s)	Platform(s)
Reconstruction	CoSAMP IHT EM+TV SART+TV	MatLab MatLab MatLab, C++ MatLab	Single-thread Single-thread Single/multi-thread Single-thread
Image restoration (denoising, deblurring)	Rician denoising Poisson denoising Poisson/Rician denoising and deblurring	MatLab, C, CnC, Cuda MatLab, C MatLab, C, Cuda	Single-thread, GPU, FPGA Single-thread Single-thread, GPU, FPGA
Registration	Fluid (non-rigid) registration	C++, CnC	Single/multi-thread, GPU, FPGA
Segmentation	Geodesic active contours Two-phase active contours	C++ C++, CnC	Single-thread Single/multi-thread GPU, FPGA

RICE

Raw data acquisition

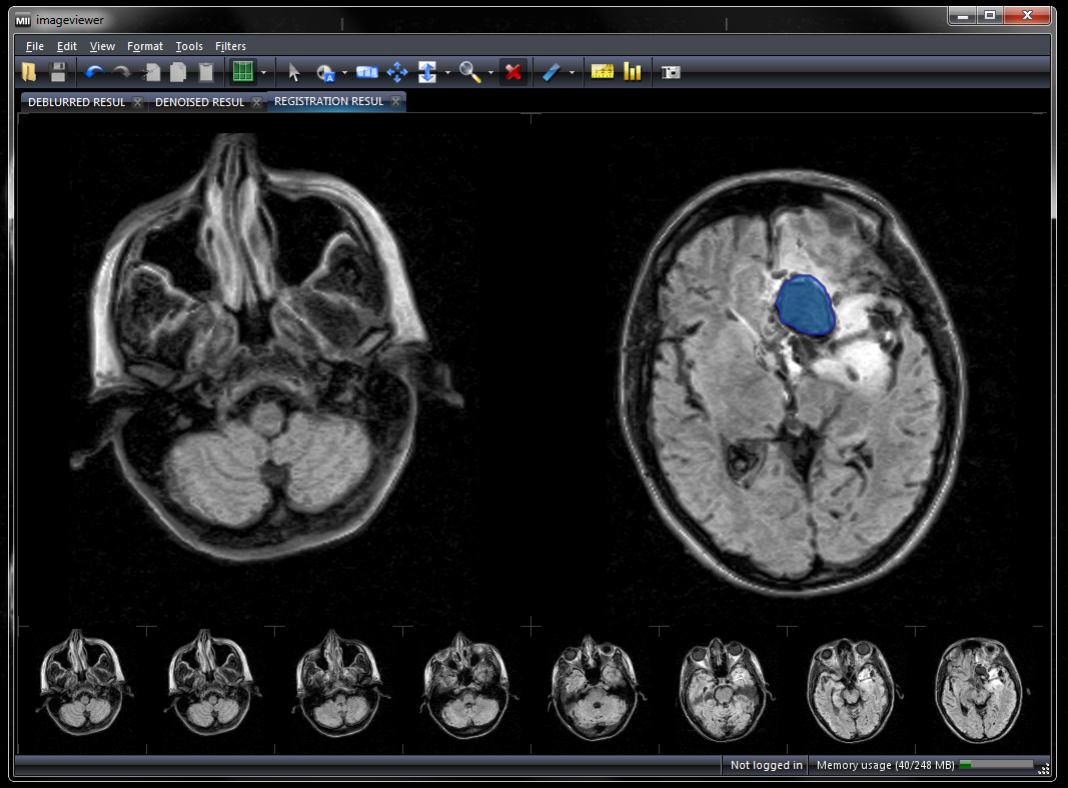
Reconstruction

Image restoration (denoising, deblurring)

Registration

Segmentation

Analysis

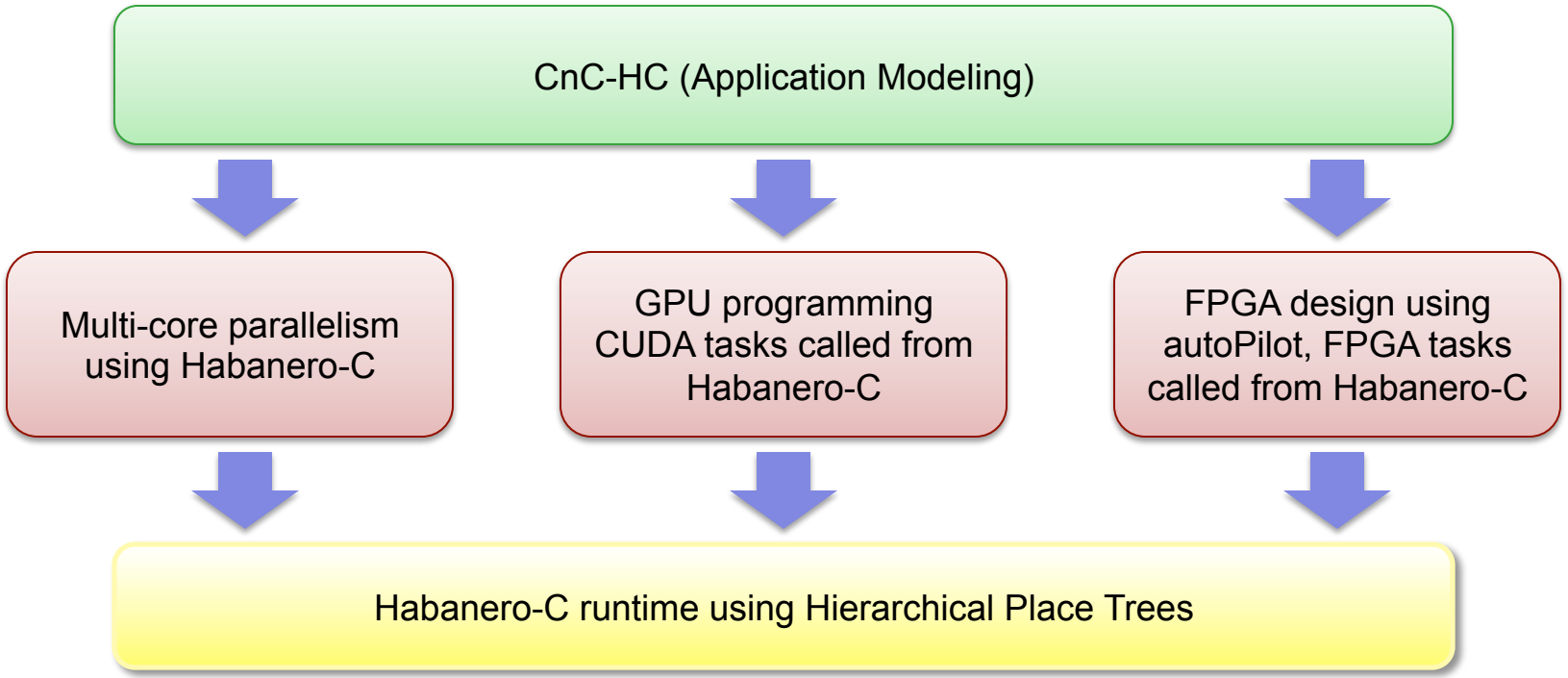


Outline

- Domain-Specific Computation
- Medical Imaging Pipeline
- **CnC Model of the Medical Imaging Pipeline**
- Locality and Heterogeneity: Hierarchical Place Trees
- Experimental Results and Conclusions

RICE

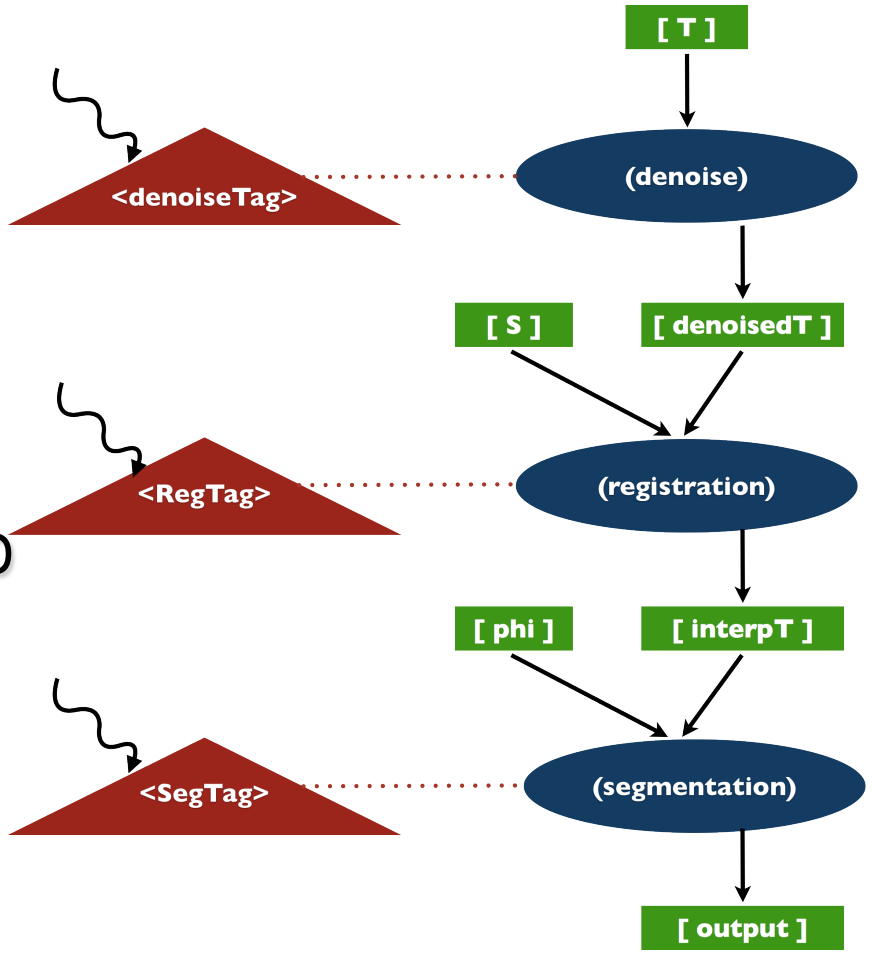
Toolchain



RICE

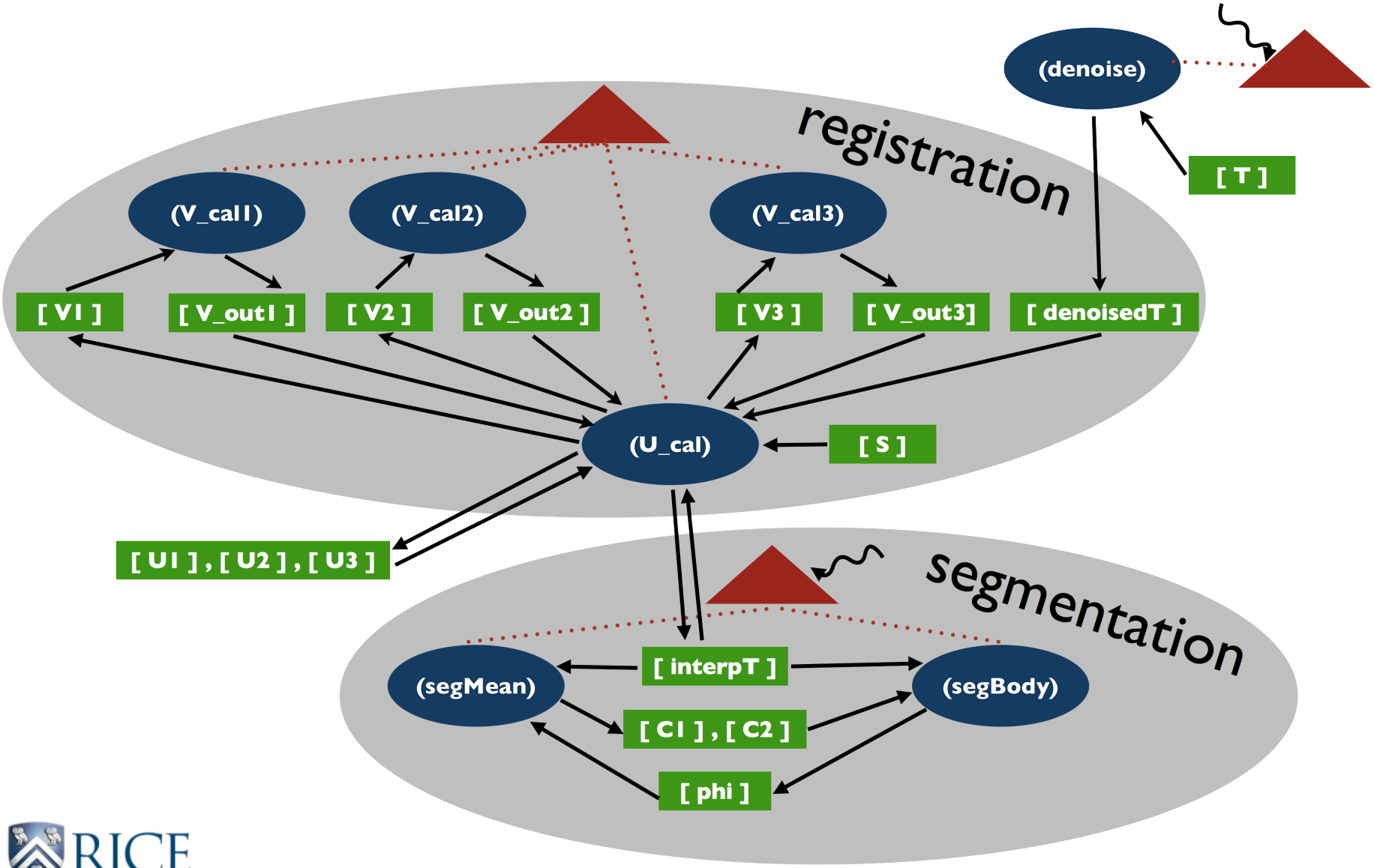
Why CnC for Modeling?

- Specify only the semantic ordering requirements
  - Easier and depends only on application
  - Separation of concerns
- Application modeling is similar to drawing on a white board
- Reuse the CnC model for mapping



RICE

Coarse-Grained CnC Graph for the Image Pipeline

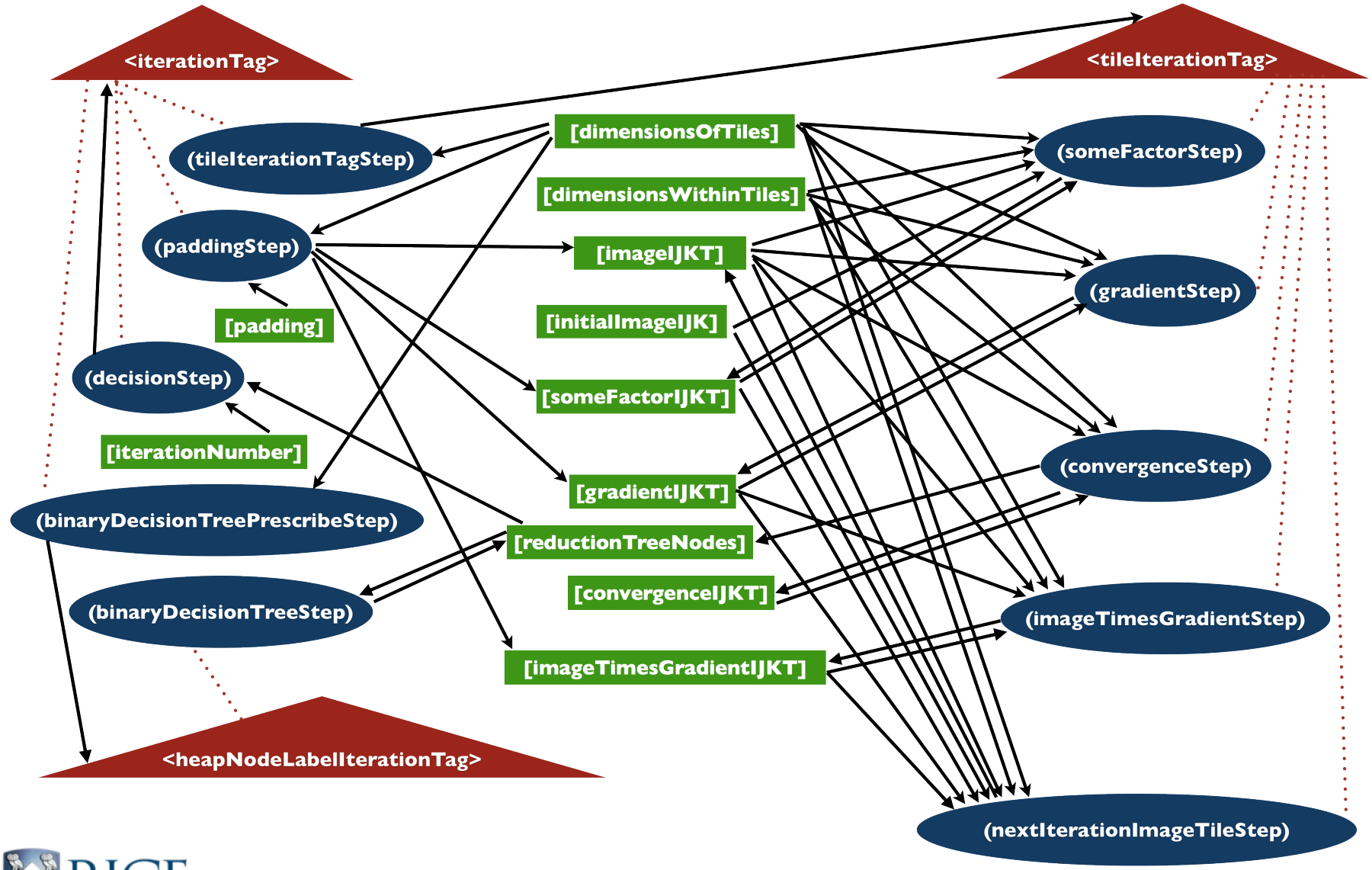


RICE

Lessons Learned: Registration and Segmentation

- CnC is great for coarse-grained modeling
- Hierarchy would help a lot in the modeling phase
  - Right now, we have multiple versions of the same CnC code
- Memory management an issue
  - Still have to resort to "cheating" (violating DSA)
  - Relatively simple problem, get counts and/or DSA space folding would solve it
- Habanero-C still a more "natural" choice for expressing fine-grained, regular parallelism
  - Parallel loops inside CnC steps implemented in HC

Fine-Grained CnC Graph for the 3D Denoise



RICE

Lessons Learned: Rician Denoising

- Lack of reductions
  - Convergence checking is an AND-Reduction that is hardcoded
- Non-native iteration-space description
  - 2D Tiling increases tuple sizes to 5
  - Non intuitive coding of time dimension
- Tag function restrictions for data-driven execution
  - 5-stencil computation needs padding if step code doesn't change
  - Or every base condition has to be a separate step implementation

Outline

- Domain-Specific Computation
- Medical Imaging Pipeline
- CnC Model of the Medical Imaging Pipeline
- **Locality and Heterogeneity: Hierarchical Place Trees**
- Experimental Results and Conclusions

RICE

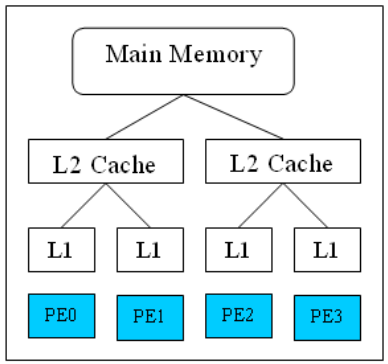
Implementing Application Steps using Habanero-C

- Extension of C language with support of async-finish lightweight task parallelism
  - Principle is similar to X10 and Habanero Java
  - Lower-level compared to CnC
    - CnC does dependency tracking; HC requires manual dependency control between async tasks
  - More suitable for loop-level parallelism with in-place updates
  - Coprocessor invocation can also be done from HC

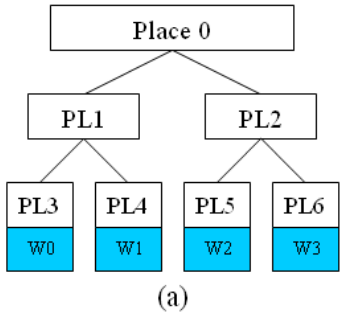
RICE

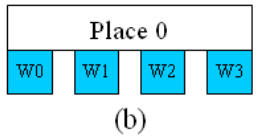
Hierarchical Place Trees (HPT)

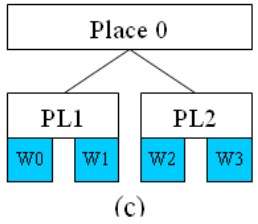
- Past approaches
  - Flat single-level partition e.g., HPF, PGAS
  - Hierarchical memory model with static parallelism e.g., Sequoia
- HPT approach
  - Hierarchical memory + Dynamic parallelism
- Place represents a memory hierarchy level
  - Cache, SDRAM, device memory, …
- Leaf places include worker threads
  - e.g., W0, W1, W2, W3
- Places can be used for CPUs and accelerators
- Multiple HPT configurations
  - For same hardware and programs
  - Trade-off between locality and load-balance



A Quad-core workstation







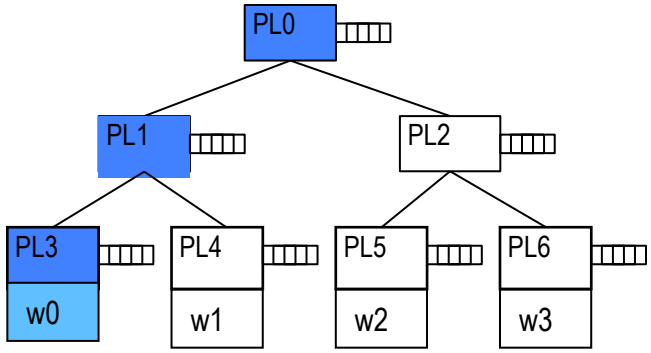
Three different HPT's
for a quad-core processor

"Hierarchical Place Trees: A Portable Abstraction for Task Parallelism and Data Movement", Y.Yan et al, LCPC 2009

RICE

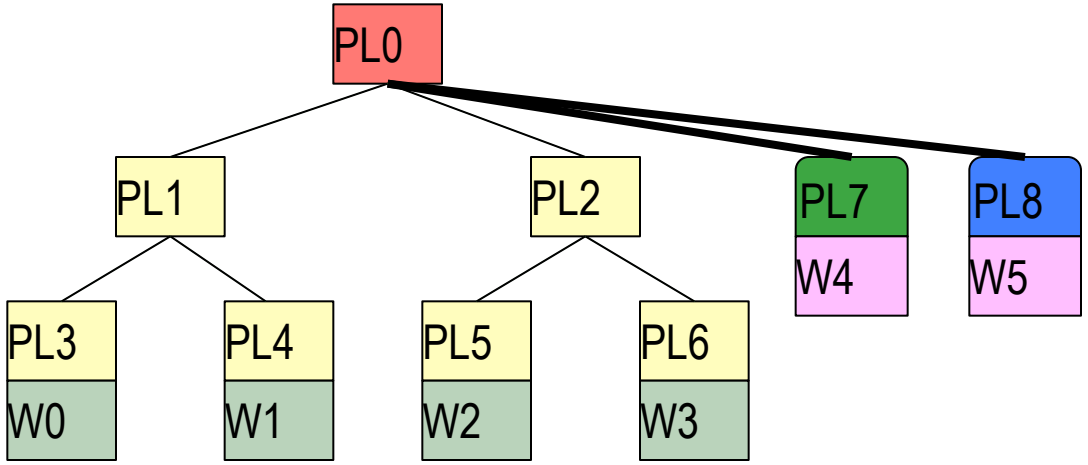
Locality-aware Scheduling using the HPT

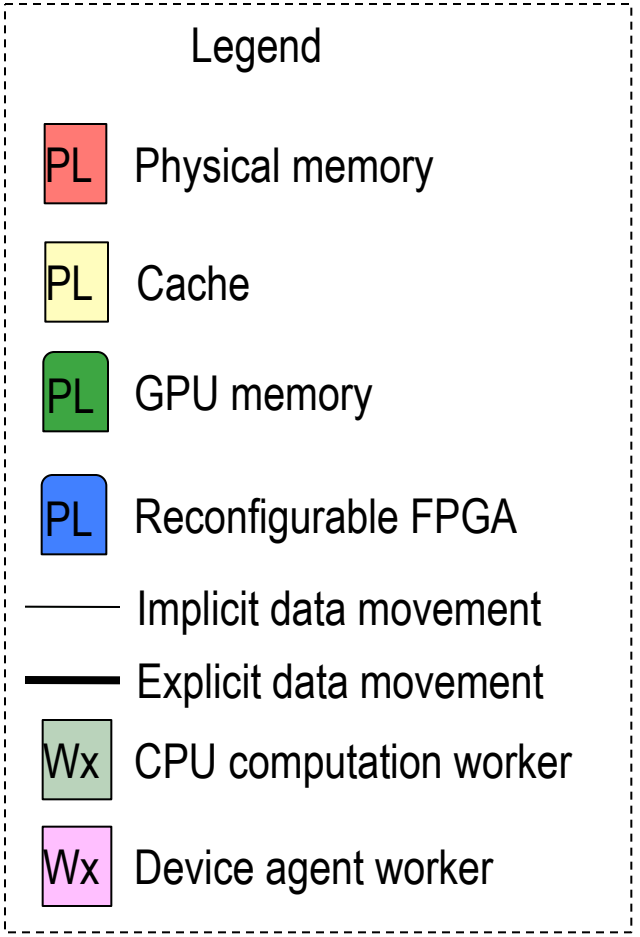
- **Workers attached to leaf places**
  - **Bind to hardware core**
- **Each place has a queue**
  - **async at(*pl*) <*stmt*>: push task onto *pl*'s queue**



- A worker executes tasks from ancestor places
  - W0 executes tasks from PL3, PL1, PL0
- Tasks in a place queue can be executed by all workers in the place's subtree
  - Task in PL2 can be executed by workers W2 or W3

Adding Heterogeneity to HPT



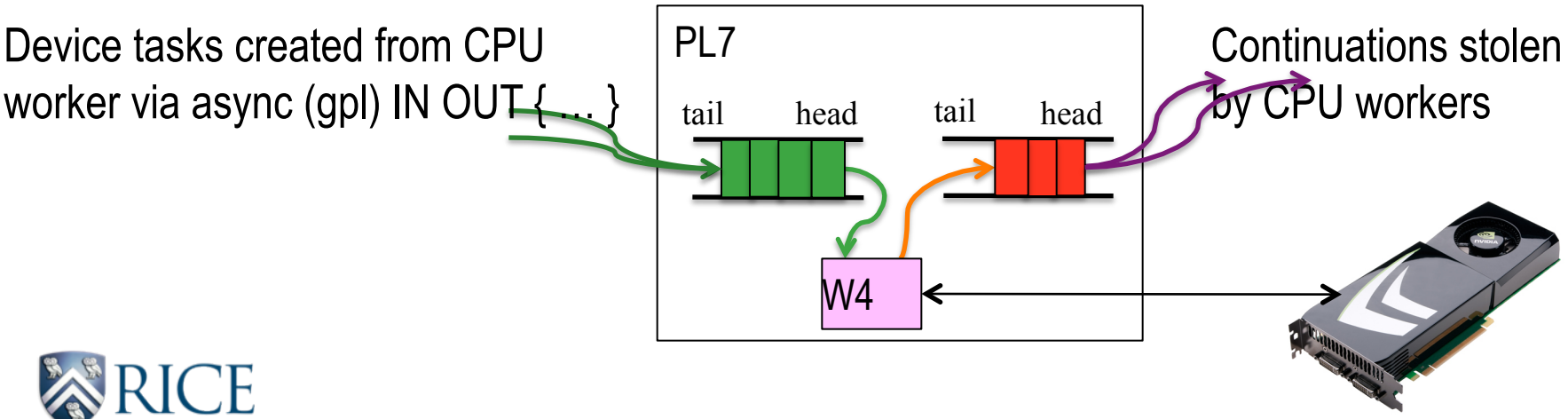


- **Devices (GPU or FPGA) are represented as memory module places and agent workers**
  - GPU memory configurations are fixed, while FPGA memory is reconfigurable at runtime
- **Explicit data transfer between main memory and device memory**
  - Programmer may still enjoy implicit data copy between them
- **Device agent workers**
  - Perform asynchronous data copy and task launching for device
  - Lightweight, event-based, and time-sharing with CPU

RICE

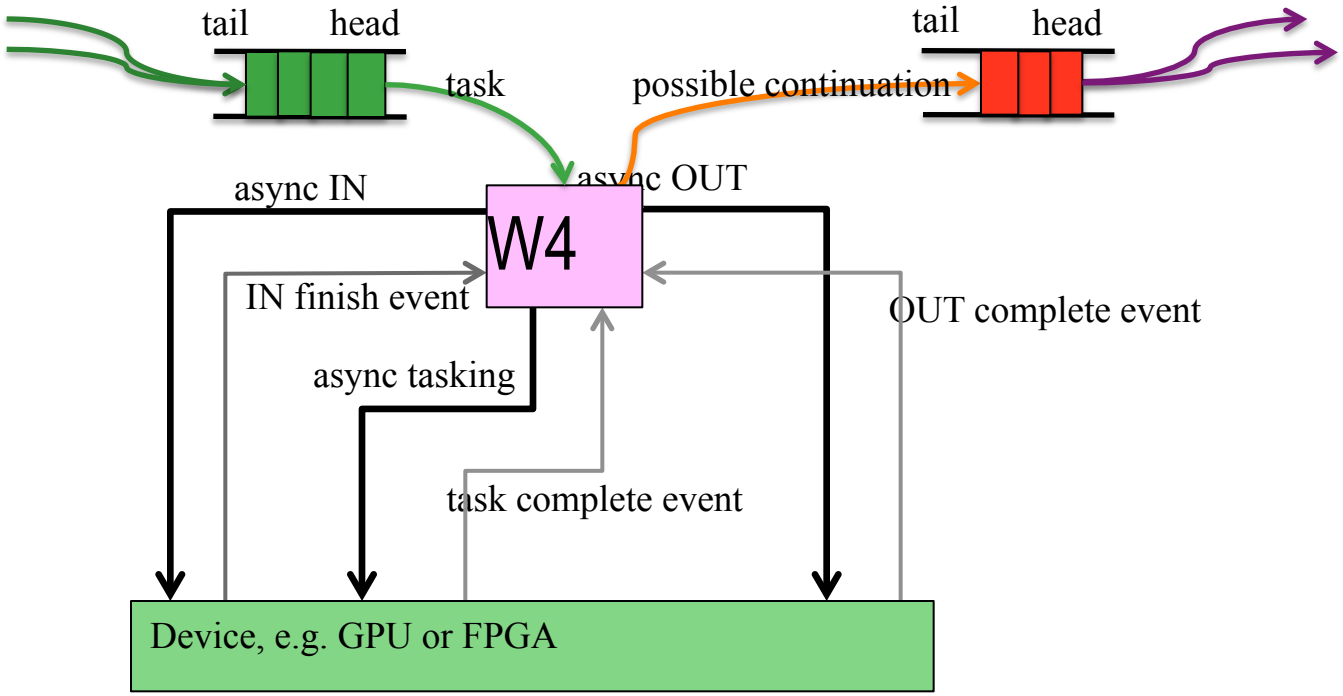
Hybrid scheduling

- Device place has two HC (half-concurrent) mailboxes: inbox (green) and outbox (red)
  - No locks – highly efficient
- Inbox maintains asynchronous device tasks (with IN/OUT)
  - Concurrent enqueuing device tasks by CPU workers from tail
  - Sequential dequeuing tasks by device agent workers from head
- Outbox maintains continuation of the finish scope of tasks
  - Sequential enqueuing continuation by agent workers
  - Concurrent dequeuing (steal) by CPU workers



Asynchronous data copy and task execution

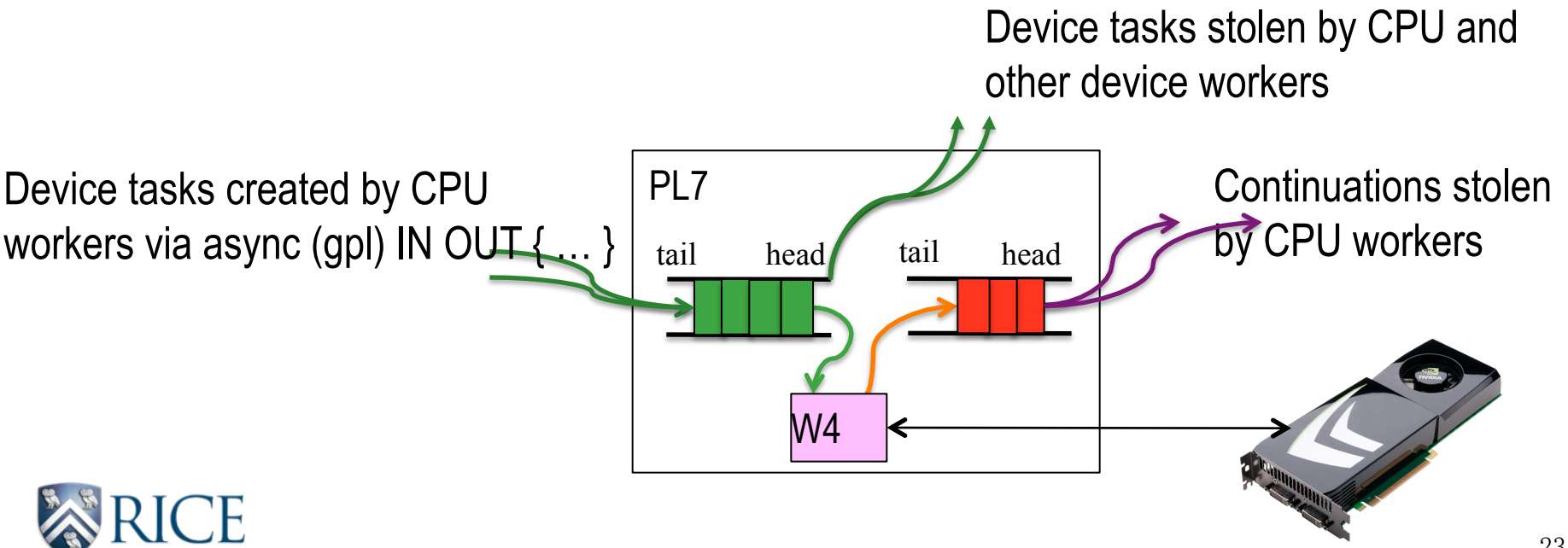
- Three asynchronous stages of each device tasks
  - Data copy-in, task launching, data copy-out
  - They all can overlap for different tasks; data copy utilizes hardware DMAs
- Lightweight event-based agent workers
  - No blocking on any of the three stages
  - Zero-contention to access both inbox and outbox
- Can be implemented in hardware!



RICE

Cross-Platform Work Stealing

- **Steps are compiled for execution on CPU, GPU or FPGA**
  - **Same-source multiple-target compilation in future**
- **Device inbox is now a concurrent queue and tasks can be stolen by CPU or other device workers**
  - **Multitasks, range stealing and range merging in future**



RICE

Support for heterogeneous execution in CnC

- GPU steps can be specified in CnC using a {step} syntax
  - The translator generates appropriate async at (gpu_place) calls
- Ranges (t1..t2) are useful for specifying sets of steps to be executed on GPU
- CnC-HC requires the tags of input items to be a function of step tags
  - Simplifies the scheduling since we only create a device task when all its input data is available
- Similar approach can be done for FPGA step specification

RICE

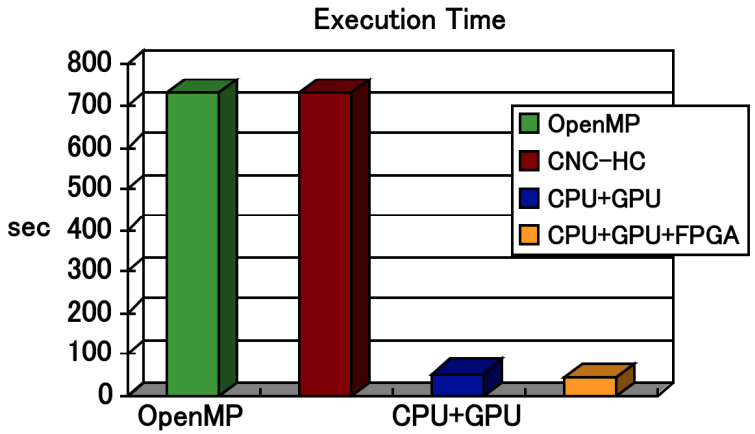
Outline

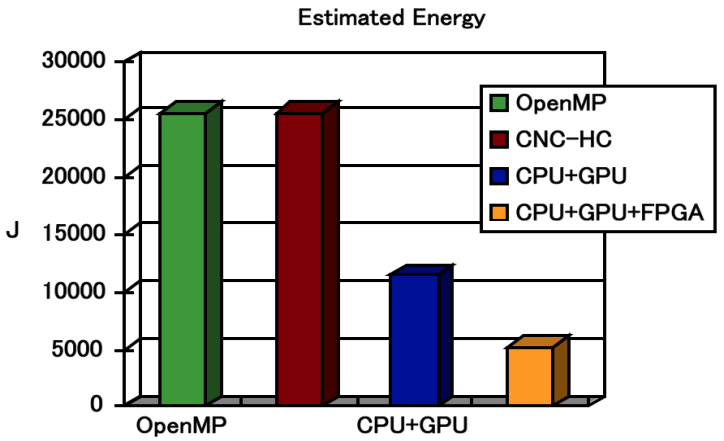
- Domain-Specific Computation
- Medical Imaging Pipeline
- CnC Model of the Medical Imaging Pipeline
- Locality and Heterogeneity: Hierarchical Place Trees
- **Experimental Results and Conclusions**

RICE

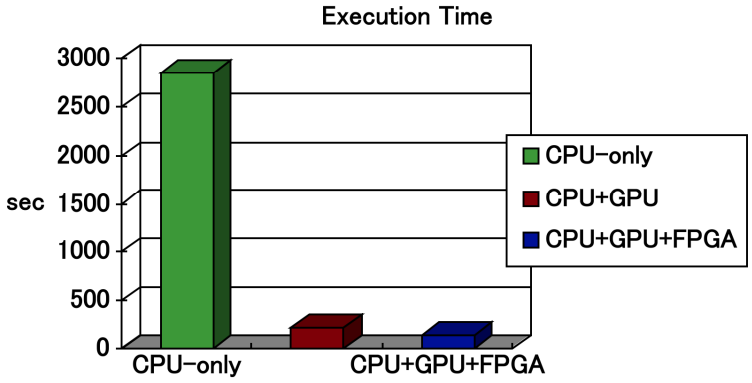
Experimental Results

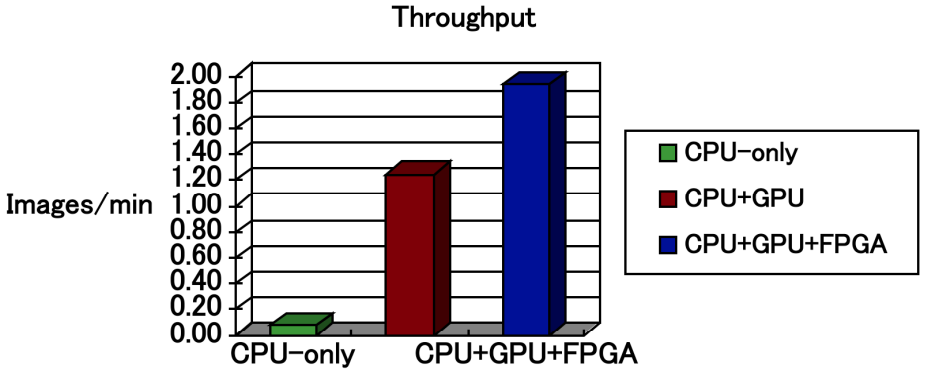
Pipeline: Denoise-->Registration (200 iterations)-->Segmentation (100 iterations)





Multi-images (4 images)





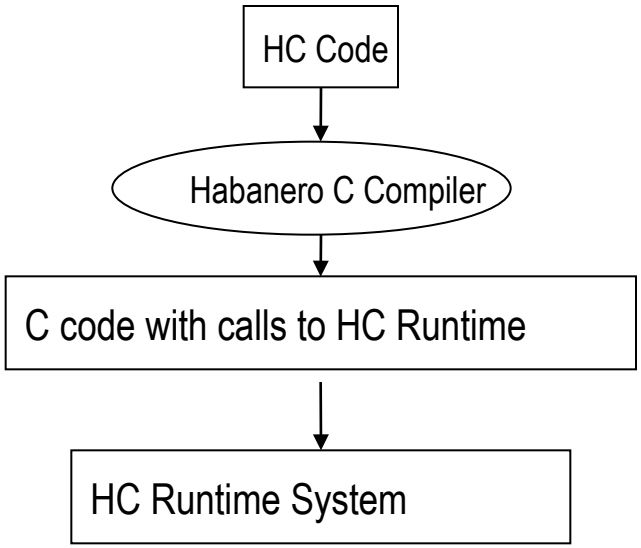
Conclusions and Future Work

- CnC is very suitable for domain-specific application modeling
  - Hierarchy, reductions, better iteration space description would make it even better
- With an efficient runtime and translator implementation, CnC can lead to a very efficient application mapping onto heterogeneous, customizable platforms
  - Cross-platform work stealing, load balancing vs. data movement, memory management

RICE

Habanero-C Compiler

- **Habanero C compiler (source-to-source)**
  - **AST nodes and parser**
  - Traversal pass: **Canonicalization of function calls**
  - Traversal pass: **mark suspendable functions**
  - Traversal pass: **build function frame** (optimization)
  - Transformation: **async, finish, and suspendable functions**

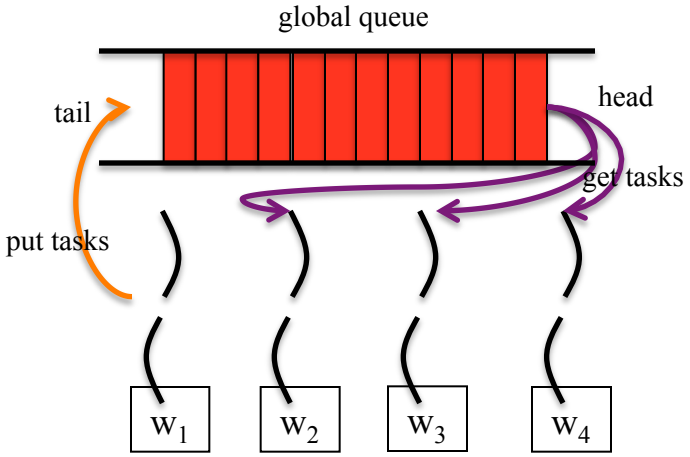


- **Example Habanero-C program**

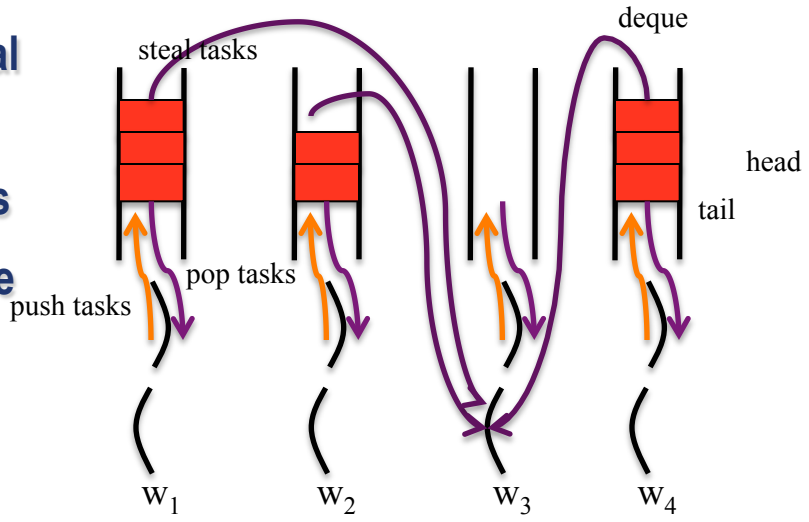
```
finish {
    /* launch GPU partitions on ngpus GPUs */
    for (i=0; i<ngpus; i++) {
        async at (gpu_pls[i]) in(A_part, B_part, part_size) out(C_part, part_size) {
            vecadd_gpu(A_part, B_part, C_part, part_size); } // C[*] = A[*] + B[*]
    } // for
    . . .
} // finish
```

Habanero-C runtime: Scheduling Paradigms

- **Work-Sharing (X10 v1.5, OpenMP, …)**
  - Busy worker re-distributes the task eagerly
  - Global thread/task/team queue
  - Access to the global queue needs to be synchronized: scalability bottleneck
- **Work-Stealing (Cilk, TBB, …)**
  - Distributed task pools: Each worker has a local double-ended queue (deque)
  - Idle worker steals the tasks from busy workers
  - Busy worker pays little overhead just to enable stealing
  - Better scalability



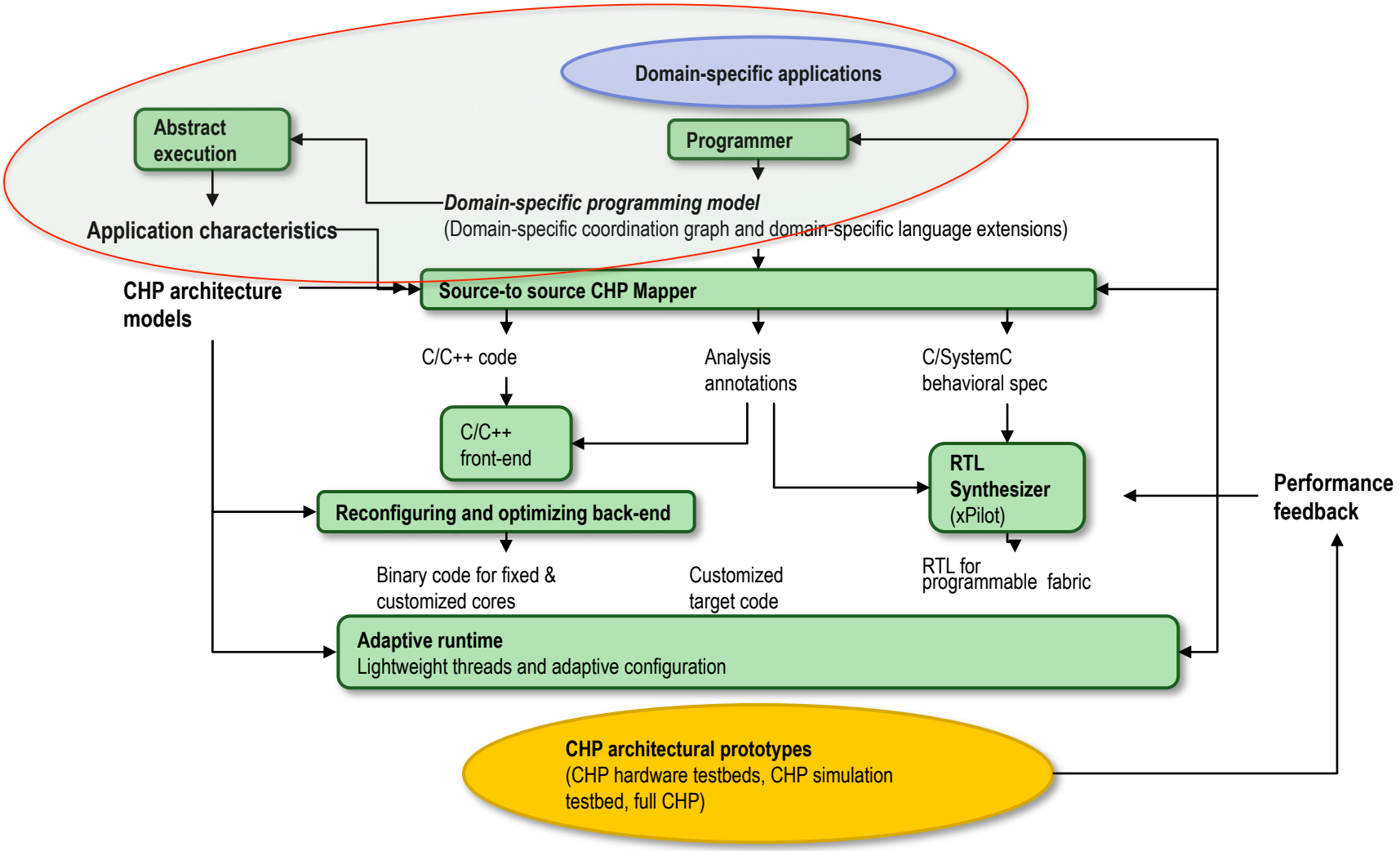
work-sharing



work-stealing

RICE

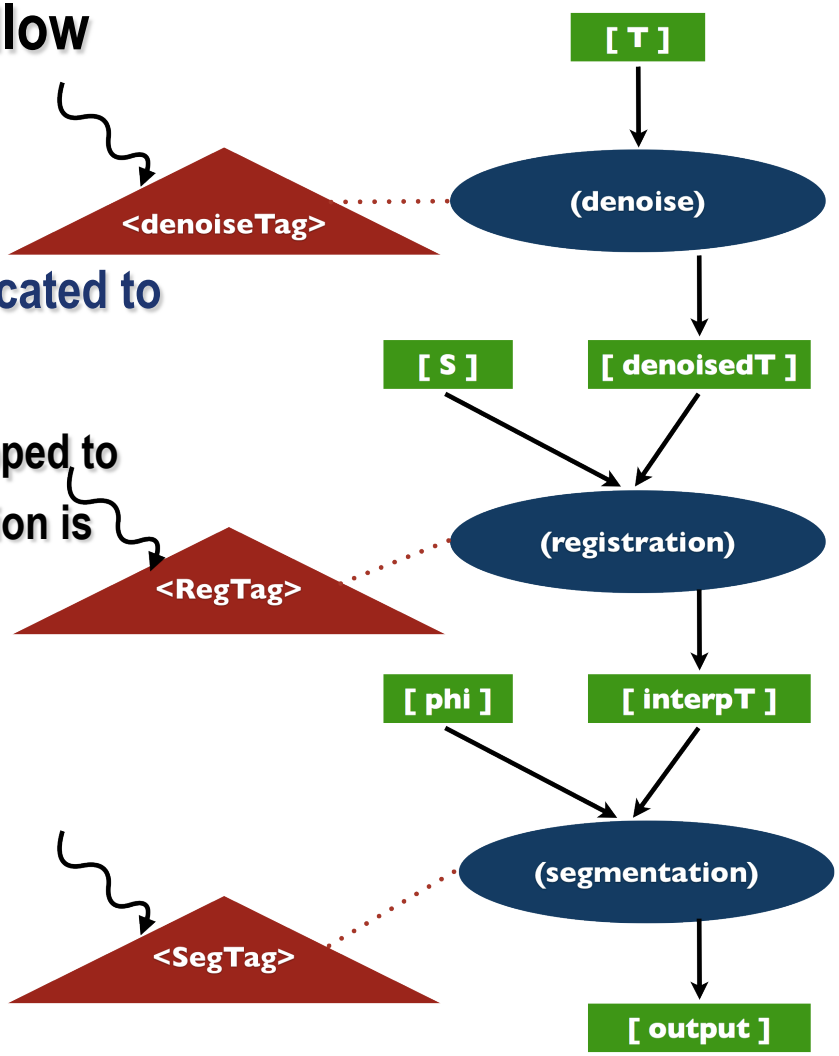
CHP Modeling Role



Coprocessor Invocation with Multi-Threading

- **Problem: the coprocessor may not allow two simultaneous coprocessor calls**
  - Use HC places (and HPT) to enqueue the coprocessor calls to a queue that is dedicated to each coprocessor
- **Example: a simple pipeline where denoise is mapped to CPU, registration is mapped to FPGA, segmentation is mapped to GPU**

```
place_t **fpga_pls=(place_t**) malloc(sizeof(place_t*));
    hc_get_places(fpga_pls,FPGA_PLACE);
    finish {
        place_t* pl=fpga_pls[0];
        async(pl) IN(denoisedT0,S1,interpT_float_h,m,n,p){
            REG_fpga(denoisedT0,S1,interpT_float_h,m,n,p);
        }
    }
```



RICE