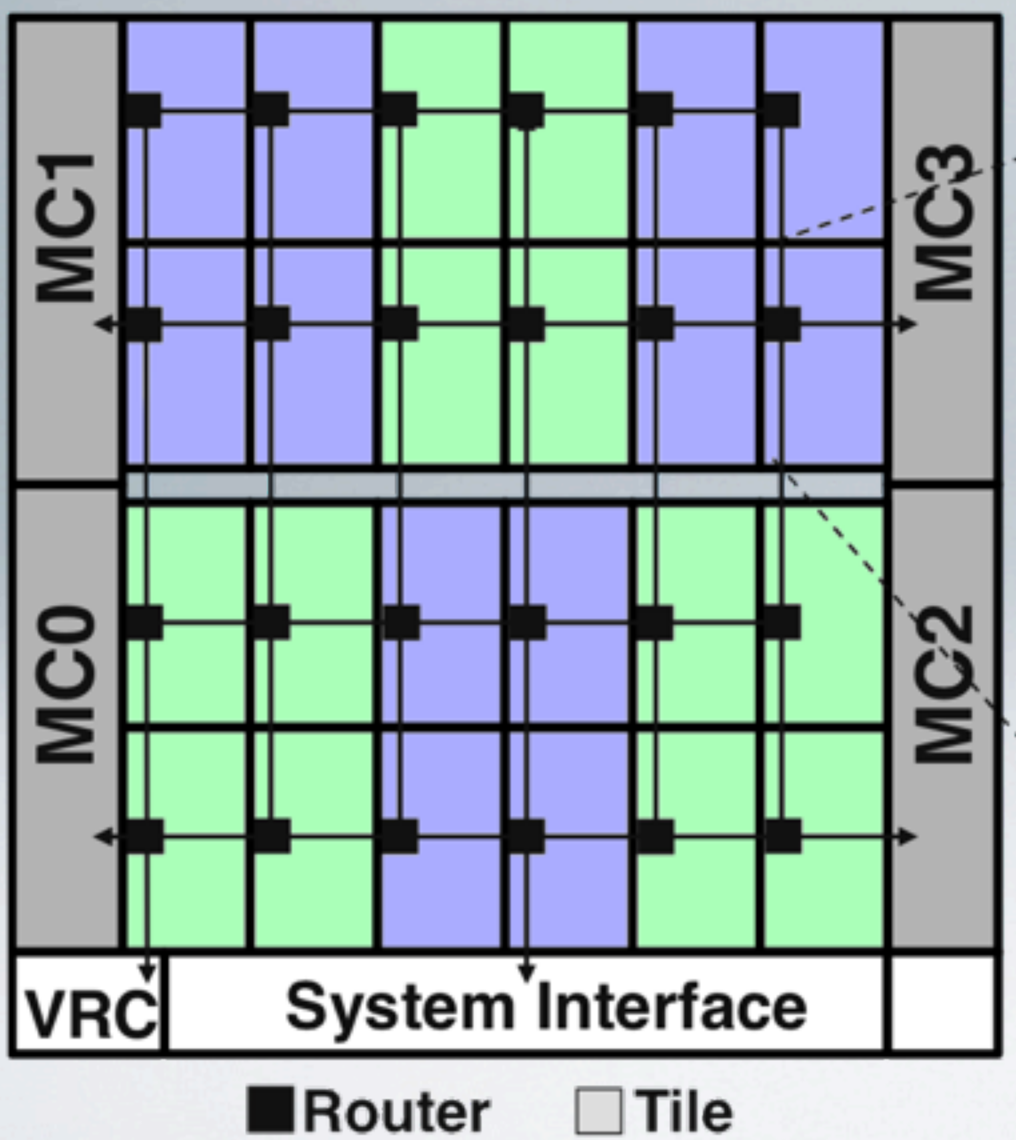
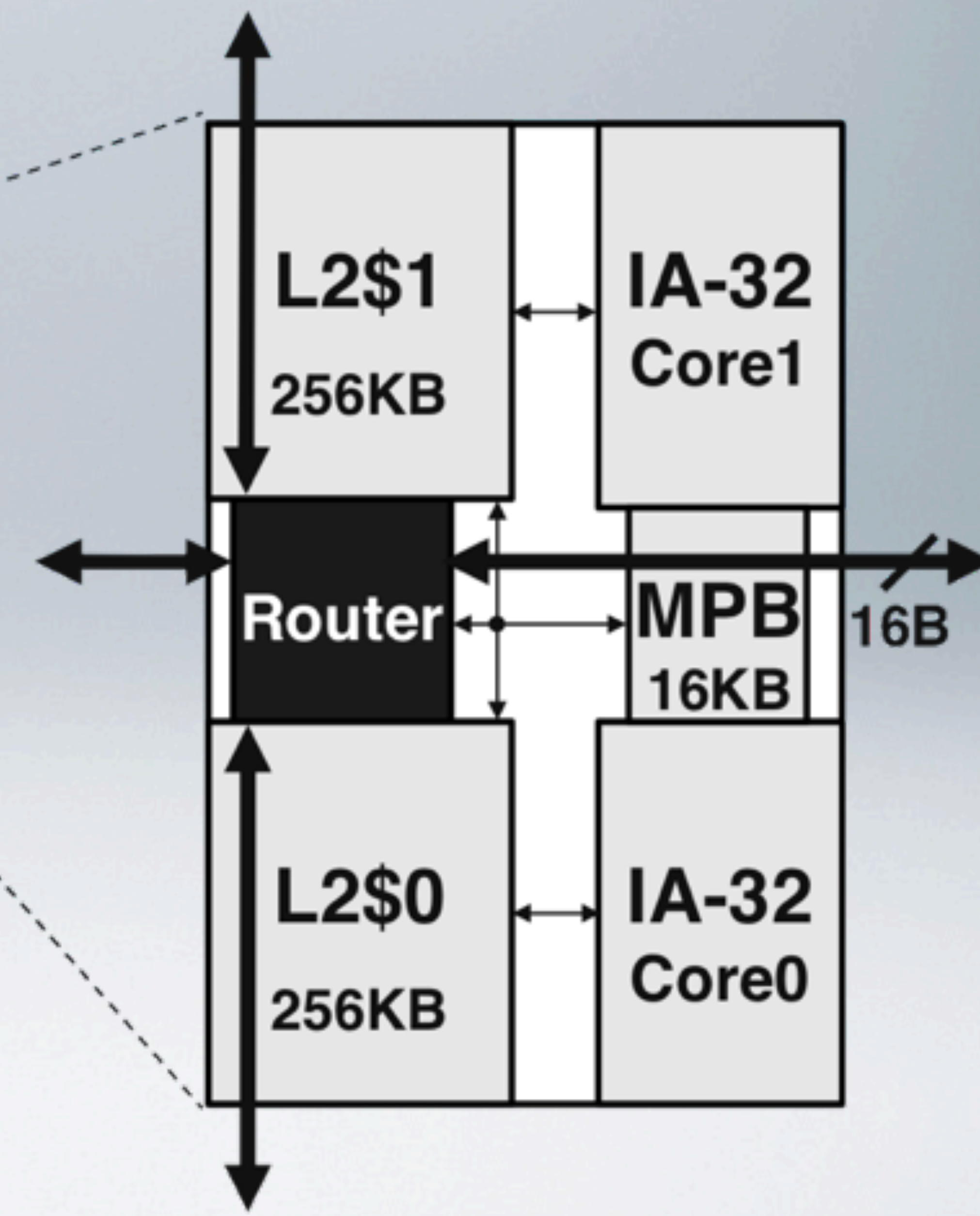
# TOWARDS SCALABLE IMPLICIT COMMUNICATION AND SYNCHRONIZATION

Raphael Poss
Chris Jesshope
Universiteit van Amsterdam

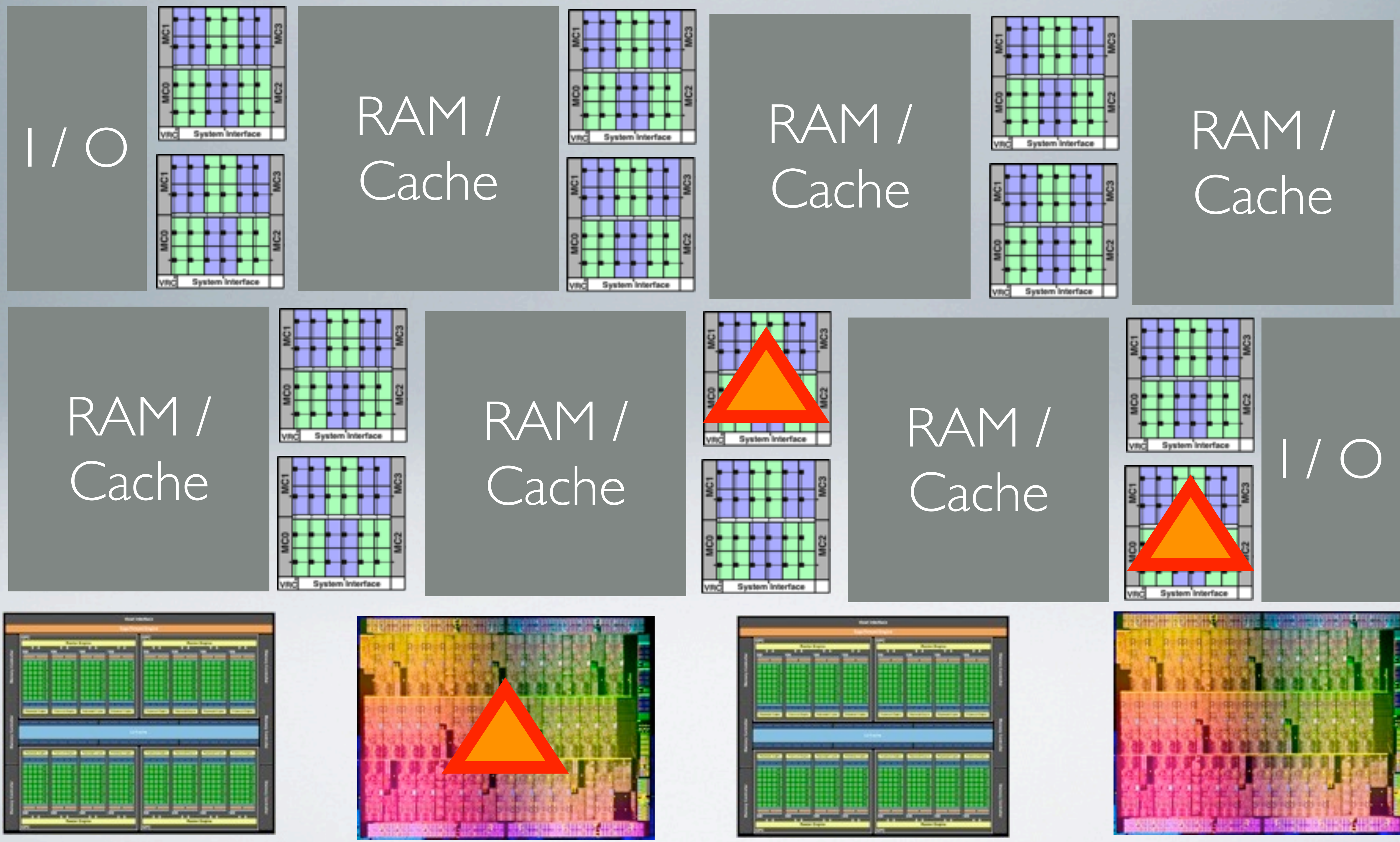
AMP'10, Toronto, June 2010

# INTEL'S SINGLE-CHIP CLOUD



# LARGER SYSTEMS




(NVIDIA's GP100) (Intel Knight's corner)

# FUTURE COMPUTING SYSTEMS

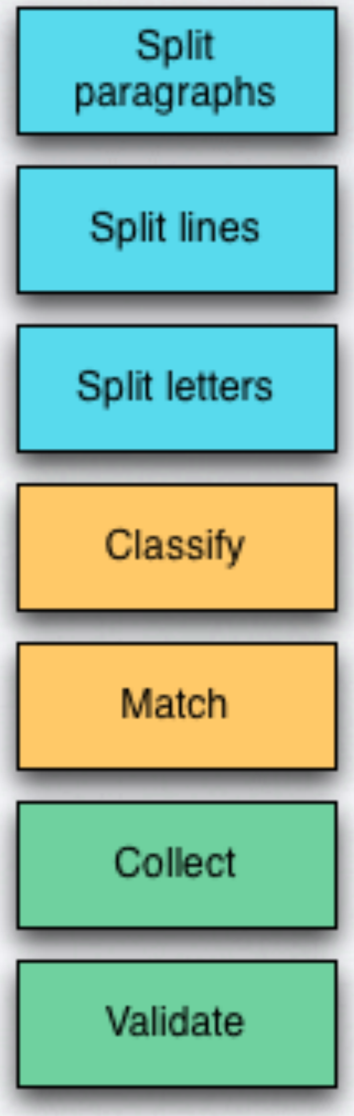
- *Scale:* now 100s of cores, tomorrow 100.000s and more
- *Space heterogeneity*: general-purpose vs. specialized, different ISAs, heterogeneity in primitives (communication, synchr.)
- *Time heterogeneity*: varying characteristics over time; mapping, routing, distribution and time scheduling become dynamic
- Synchronization and all forms of *non-local knowledge propagation* have a non-negligible cost

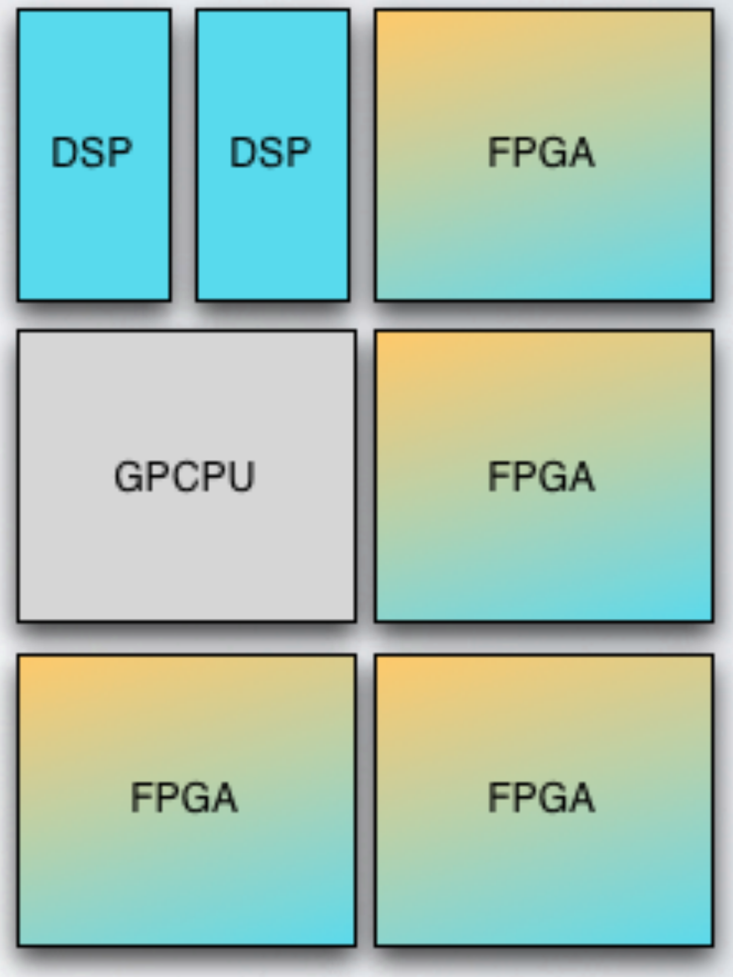
# THE CHALLENGE OF HETEROGENEITY

- **Granularity mismatches**:
  - Between levels, now distinct programming methodologies
    eg. CUDA vs. MPI, compiler-driven vectorization vs. separate coordination
  - Dynamic unfolding of concurrency over dynamically evolving granularities, *re-clustering* must be automatic and *fast*
  - Heterogeneity in algorithm representations does not scale!
    ... unless *automatically generated* from a common origin

# USE CASE : OCR

- 2 DSPs, 4 FPGAs, 1 GPCPU
- 2 APIs, 4 ISAs, 3 data layouts
- 3 concurrency granularities
- **Desired**:
  ***one program* per algorithm**
  \+ *specialization* to targets



# STRATEGY DURING PROGRAM WRITING

- **Provide information but leave flexibility to the environment (compiler, rt, OS, hw)**
  - Provide/use constructs that *separate concerns*:
    *concurrency* vs. scheduling, *data dependencies* vs. communication
  - Provide/use *specializable semantics*, express *patterns*
    *re-scheduling* of code paths, *aggregatable* communication patterns
  - Provide/use handles to *scope synchronization*
    both for *precedence* and *exclusion*

# STRATEGY FROM THE RUNTIME SIDE

- **Use the information, transform when necessary:**
  - Pick *resources on-demand* upon concurrency, *reconfigure* (expression of concurrency must be resource agnostic)
  - Use information over synchronization scopes and data dependency endpoints to *specialize network routing* (they must be derivable from programs automatically)
  - *Tolerate granularity mismatches* at run-time by *specialization* (language semantics must allow this) — not API abstractions!

# EXAMPLES: SPECIALIZABILITY (DIVERSITY OF PRIMITIVES)

## Message passing

grab A, B, channel x, y
p1 = delegate@A { send(y, f(recv(x))) }
p2 = delegate@B { send(y, g(recv(x))) }
send(x, u); a = recv(y) + recv(y);

hard

## Shared memory

alloc x, y;
p1 = fork(f, u, &x);
p2 = fork(g, u, &y);
join(p1); join(p2); a = x + y;

simpler

## Specializable concurrency

a = **async** f(u) + **async** g(u)

simpler

# EXAMPLES: SPECIALIZABILITY (REDUCING CONCURRENCY)

| Specializable code | Reduced concurrency |
|---|---|
| **parallel** for(i in s) do(i) | for(i in s) do(i) |
| a = **async** f(u) + **async** g(v) | a = f(u) + g(v) |
| f(s, x) { **critical** upd(s) }<br>f(s, a) **\|** f(s, b) | f(s, x) { upd(s) }<br>f(s, a) ; f(s, b) |

# EXAMPLE: SCOPING EXCLUSION

<table>
<tr><th>Without scope</th><th>With scope</th></tr>
<tr><td>f(s, x) {<br>    ...<br>    critical {<br>    upd(s, x);<br>    }<br>    ...<br>}<br>... f(s, a) | f(s, b) ...</td><td>f(state s, x) {<br>    ...<br>    exclusive_with(s) {<br>        upd(s, x);<br>    }<br>    ...<br>}<br>... f(a) | f(b) : sharing (s) ...</td></tr>
<tr><td>No information about affinity between asynchronous processes</td><td>Information about affinity is provided at the point concurrency is created</td></tr>
</table>

# NEW CHALLENGES

- P < N vs. N > P: how to recognize? Need formal systems to *describe* heterogeneous resources and dynamic concurrency, and evaluate bindings *at multiple levels of granularities*
- Specialization: how, who and when? Cooperation between *compilers, concurrency runtimes, operating systems, hardware*
- Expressivity: how to use implicit constructs and still provide enough information for efficient scheduling and specialization? *Fine-grained* data dependencies and synchronization scopes

# RESEARCH DIRECTIONS

- *Extend* languages and determine *best practices* to propagate more knowledge from programs to infrastructure; focus on:
  - functional languages (SAC, Haskell), dataflow (Cilk, SVP)
  - separate coordination vs. computation (S-NET)
- Use this knowledge and *combine* efficient space scheduling (for P > N) with *specializability* (for N > P)

THANK YOU.

# EXTRA SLIDES (COMPLEMENTS)

# CONCURRENCY OVERHEADS

| Concurrency expressed | Resources | Cost | Overhead |
|---|---|---|---|
| A ; B ; C | P = 1 | A + B + C | 0 |
| A ; B ; C | P = 2 | A + B + C | 1P unused |
| (A \| B) ; C | P = 2 | m(A, B) + s + C | s |
| (A \| B) ; C | P = 1 | A+c+B+c+C | c+c |

# COMMUNICATION OVERHEADS

| Communication expressed | Resources | Cost | Overhead |
|---|---|---|---|
| A(w.x) ; B(r.x) | P = 1 | A + B + d | d |
| A(w>x) \| B (r<x) | P = 2<br>L = 1 | m(A, B) + L + s | L+s |
| A(w>x) \| B (r<x) | P = 1<br>L = 1 | A + c + B + L | c+L |
| A(w>x) \| B (r<x) | P = 2<br>L = 0 | A + c + B + d* | c+d*<br>1P unused |

# TERMINOLOGY

| Concurrency | Run-time parallelism | Resource parallelism |
| --- | --- | --- |
| non-determinism with regards to the order in which events may occur | degree to which events actually occur simultaneously | amount of hw/sw support for independent processing |