

Thread-based models of concurrency

Chris Jesshope
Universiteit van Amsterdam

ÆTHER

- The ÆTHER project is investigating:
 - *Self-Adaptive Embedded Technologies for Pervasive Computing Architectures*
- What does this mean?
 - **self-adaptivity** means scheduling concurrency dynamically onto any available resources
 - **pervasive architecture** means dynamic resource availability
 - i.e. processing agents going on/off-line at run time

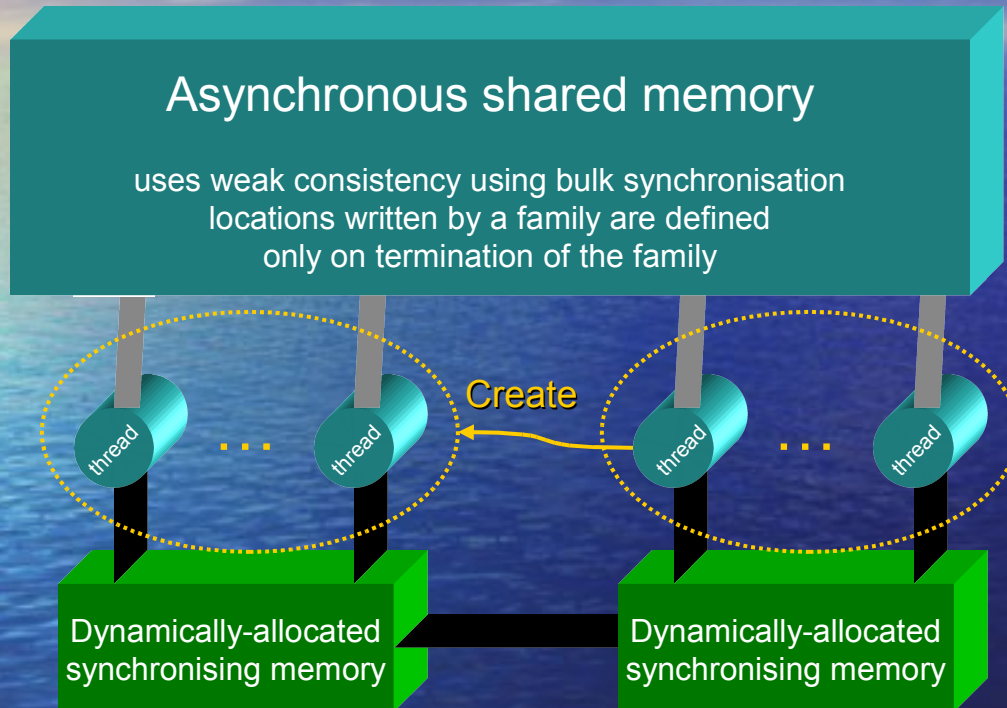
Using threads

- Thread-based models are good for this but...
 - implementations are generally slow (SW scheduling)
- The thread model needs to be implemented in HW
 - where a thread can be anything from a single instruction up to a complete application for flexibility
- Implementations need hardware for:
 - *scheduling* - KISS scheduling
 - *synchronising memory* - to drive the schedulers **here**
 - *context management* - support for concurrent contexts
 - *resource management* - where to execute threads

Microthread model

- ÆTHER will use the *microthreaded* model at the lowest levels of system interfacing
- It is designed to be implemented in HW with the following characteristics:
 - *blocking threads* - supports data-driven execution
 - *families of threads* - heterogeneous or homogeneous
 - *fine-grain synchronisation* - between threads (in a family)
 - *bulk synchronisation* - between families
 - *pre-emption* - only of families not individual threads

Microthread memory model



Shared memory can be implemented as true shared memory or local memories with communication - there are no guarantees on latency

A family of threads

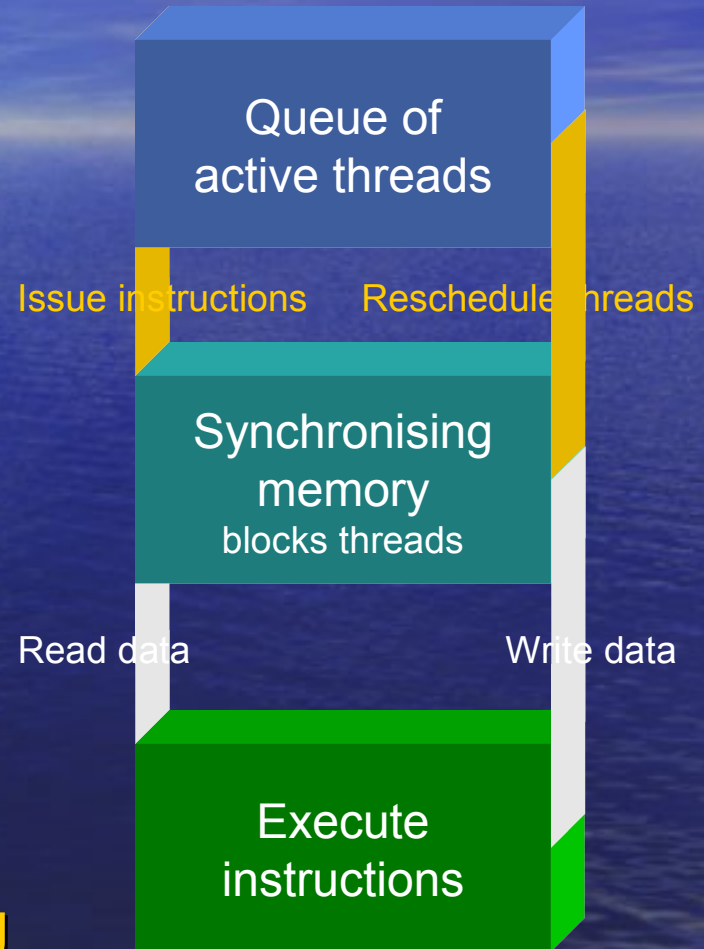
Synchronising memory would be implemented as register files or handshaking on signals

maintains dependencies between operations in concurrently executing threads and in operations on values read from shared memory

Blocking threads

- Blocking threads give a means to *tolerate latency* as long as there are many local threads
 - a thread proceeds only while its instructions have data to process
 - otherwise the thread suspends and another is executed
 - opportunities for *power conservation*
 - requires locations in synchronising memory for many suspended threads

This is data-driven dynamic scheduling



Families of threads

Threads are traditionally treated as individuals

- Families capture *regularity* and *interconnectedness* between threads and this concept helps in managing contexts and locality
 - this is a simple analogy to a loop in the sequential model
- It is the family which maps computations to resources
 - dynamically-counted threads are distributed to resources
 - it has mission goals attached - e.g. power, performance, etc.
- It is the family that supports dynamic adaptation
 - a family is parameterisable and pre-emptable
 - only a parent and its family may synchronise with each other
- It is the family that manages main memory consistency
 - a family's I/O is to asynchronous shared memory
 - family termination synchronises shared memory

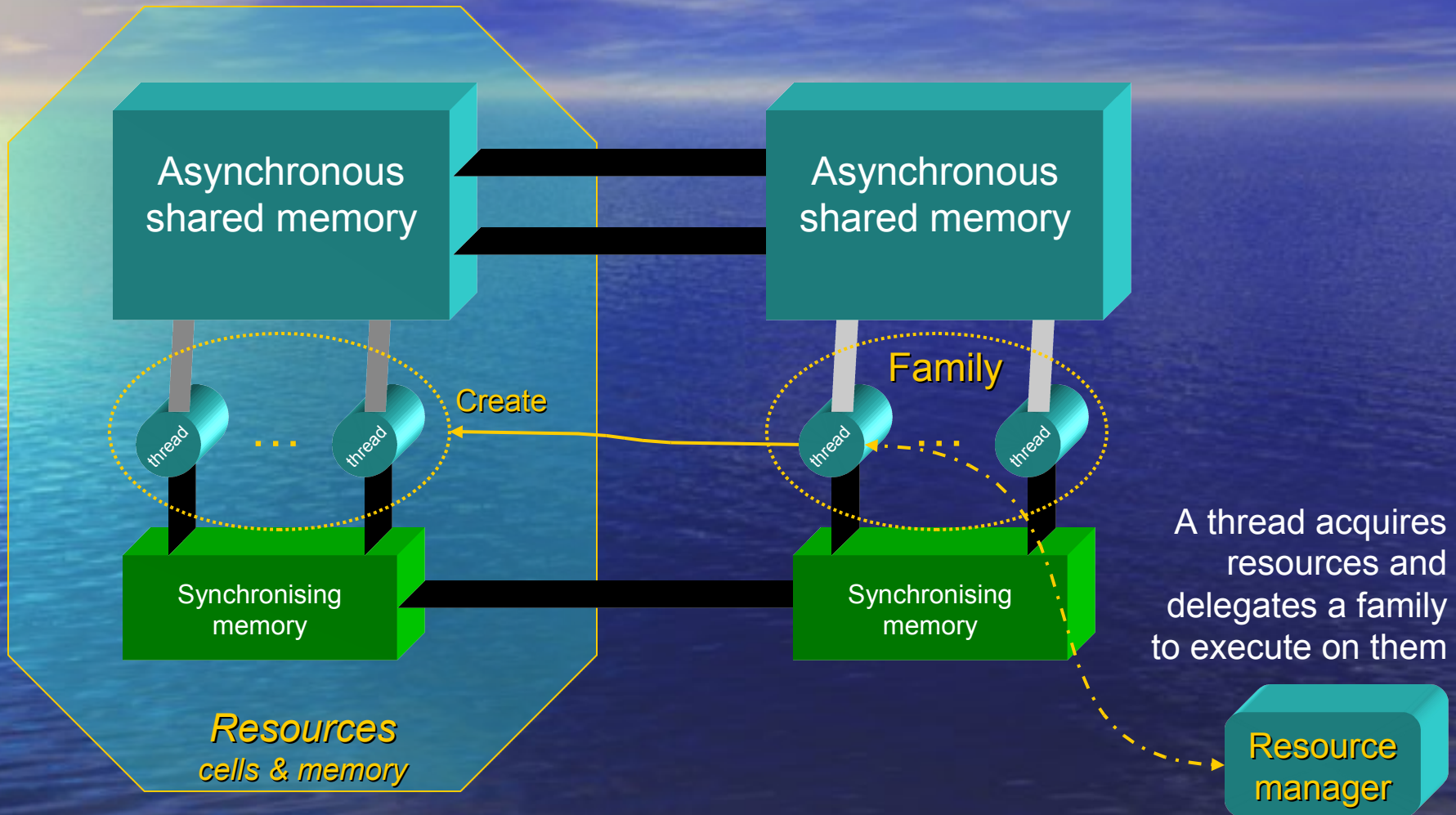
Resource management

- Resources are owned or leased by a thread for a period
 - e.g. a parent thread has exclusive access to a group of processors for the execution of its family of threads
 - resources may be FPGA cells, special functional units etc.
 - resources may also be shared between families
- Resources are acquired by negotiation with a resource manager or hierarchy of managers
- Threads relinquish resources at termination of a contract
 - may have to force a termination for delinquent threads
- Families are identified by a place and an id at that place
 - pre-emption and forced termination require collective operations on families hence the requirement for identification

Adaptation by family delegation

- Family delegation provides the high-level mechanism for adaptation
 - *this is at the software-hardware boundary*
- A thread acquires resources and delegates the execution of a family to those resources
 - this is a communication of data, code or both from the thread creating the family to the device executing it
 - It also provides synchronisation on shared memory

Family delegation - distributed



Adaptation from family implementation

- Family implementation provides the low-level mechanism for adaptation
 - *this is adaptation within the hardware platform*
- A family utilises its resources to optimise its mission goals, such as performance, power dissipation, etc. e.g.
 - executed on a conventional processor
 - executed by power-aware, data-driven μ -processors that support this thread/family model
 - executed by various paradigms on reconfigurable HW

Environments

- Delegation of the execution of a family can be performed in a different environment
 - for example from chip to chip in a system
 - in a global environment such as GRID
 - in an ambient environment with ubiquitous resources
- Protocols are required to manage acquisition, delegation and implementation for a given environment, e.g. for:
 - low-level synchronisation
 - negotiation and delegation
 - supporting a bulk-synchronised shared memory model

Tools for the thread/family model

μ TTC - microthreaded C

- Microthreaded C is a simple extension to the C language supporting this model
 - It supports constructs to:
 - **Create** families of threads
 - **Break** from within a family so that any thread can dynamically terminate its family
 - **Sync** on termination of threads in a family
 - and from an concurrent control thread
 - **Kill** or pre-empt a running family (called **Squeeze**)

μ TC Compiler

- We are developing a compiler from μ TC using gcc - *HiPEAC compiler platform 1*
 - our target is binary code for simulations of power-aware, data-driven μ -processors that support this thread/family model
 - the same front end can be used for other targets, e.g. compiling to FPGA perhaps via other languages (e.g. Handel-C)
- We are also developing a compiler from C to μ TC, so that code can be automatically parallelised for this model, using CoSy - *HiPEAC compiler platform 2*

Summary

- Thread models are good for adaptive computing
- ...but need hardware implementations where
 - threads are dynamically created and scheduled
 - threads wait for their data - perhaps indefinitely
- Families are a grouping of threads
 - families capture regularity and locality of communication
 - families have collective properties such as mission goals
 - families provide mechanisms for adaptation
 - in their implementation
 - by delegating them to dynamically acquired resources