

# Operating System Support for Asymmetric Multi-Core Architectures



Ross McIlroy, Peter Dickman and Joe Sventek  
 Department of Computing Science, University of Glasgow  
 {ross,pd,joe}@dcs.gla.ac.uk

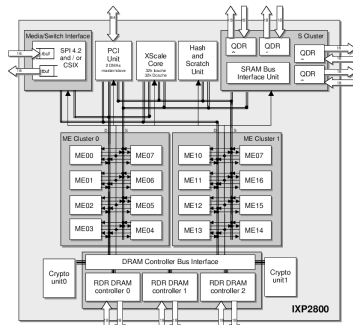


## 1: Asymmetric Multi-Core Architectures

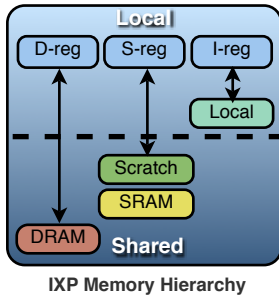
As it is becoming more difficult to extract performance improvements from single-core architectures, designers are turning to multi-core architectures.

*Asymmetric Multi-Core Architectures* can gain performance and energy saving benefits through specialisation of certain processing cores to a particular purpose.

Examples of these architectures include the Intel IXP network processor, the IBM Cell (used by the Sony PS3), and AMD Fusion.



Intel IXP network processor architecture



These architectures are notoriously difficult to program. Difficulties include:

- Inter-core communication
- Different processor instruction sets
- A complex memory hierarchy

For example, the IXP contains 4 different memory classes, each explicitly accessible with no hardware caching. Each core also contains 5 different register types and content addressable memory (CAM).

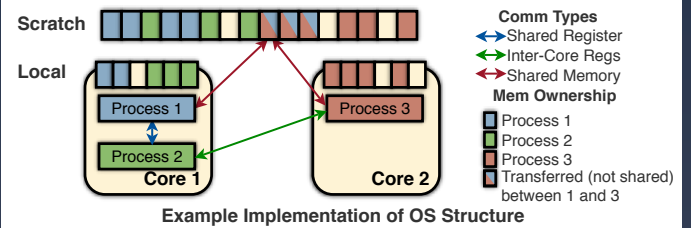
## 2: Overall OS Design

To alleviate some of these programming issues, and allow general purpose applications to make effective use of asymmetric multi-core architectures, we propose an OS structure which follows three fundamental strategies:

Strategy	Reason
Applications consist of many lightweight processes	Maximises concurrent execution of code on multiple cores
Each process sealed in a closed memory space	Enables processes to optimise their use of the complex memory hierarchy
Inter-process communication with asynchronous channels	Enables OS to choose preferred communication method at runtime

The OS and apps will be written in managed code to allow:

- Compile time code verification for memory protection and channel verification (rather than using virtual memory hardware)
- Compilation to intermediate bytecode (this abstracts the multiple instruction sets for the different core types available)



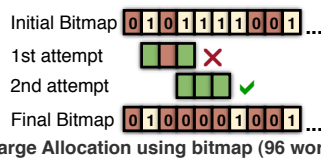
## 3: On-Core Memory Management

Managing the complex memory hierarchy of asymmetric multi-core architectures is challenging. As a first step towards our proposed OS design, we built a memory management algorithm targeted at the small on-core local memories common to these architectures.

### Challenges:

- Minimise memory overhead (only 4KB of local memory available on the IXP)
- High performance
- Minimise code footprint (max. 8K instructions on IXP)

Our algorithm splits memory into 32 blocks (of 32 words each). A bitmap (one word stored in memory) represents whether these blocks are free or in use.



Smaller free regions are stored in four free lists (for 2, 4, 8 and 16 word regions).

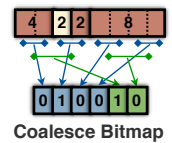
Small region allocation proceeds by:

- Round requested size up to 2, 4, 8 or 16 words
- Remove free region from the appropriate list
- If the list is empty a larger region is split



### Small Region Splitting

Regions are split in a predictable 2<sup>n</sup> manner. This enables region coalescing using a coded bitmap (stored in free memory), instead of wasteful boundary tags.



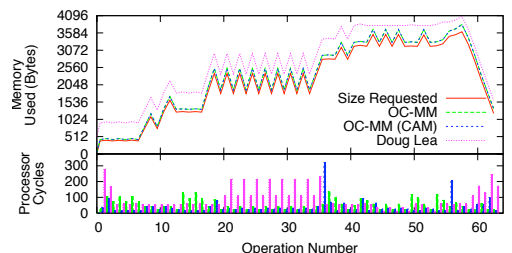
## 4: Experimental Results

We compared this algorithm against Doug Lea's malloc algorithm (naively ported to the IXP) for on-core memory management. Resource usage and performance was profiled across memory traces from multiple commonly used applications (tar, gcc, python, etc).

We also compared an enhancement to our algorithm, which defers coalescing of memory regions by caching the most recently freed regions in the IXP on-core CAM.

Resource	OC-MM	OC-MM (CAM)	Doug Lea
State memory (word)	10	10	129
CAM memory (word)	0	16	0
Code memory (word)	341	507	1634
Avg Allocation (cycle)	72.8	50.9	70.7
Allocation Success	91%	86%	77%

Resource Usage Comparison



Python Trace Performance Comparison

## 5: Conclusions and Future Work

Asymmetric multi-core architectures are difficult to program. We have presented an OS structure which may alleviate some of these issues, and have taken the first steps towards realising this OS by managing on-core local memory with minimal overheads.

Future work will include: memory management of the shared memory classes on the IXP; porting to other architectures; and implementing the OS design presented here.

## Acknowledgements

This work is funded by the Carnegie Trust for the Universities of Scotland.

With thanks to Doug Lea for open sourcing his memory allocation implementation.

