

***Tapping the Power of the Cell  
(for Video Games)***

***Mike Acton***



***For more information email:***

***[macton@highmoonstudios.com](mailto:macton@highmoonstudios.com)***

***[macton@cellperformance.com](mailto:macton@cellperformance.com)***

***See, for Cell performance tips:***

***[www.cellperformance.com](http://www.cellperformance.com)***



# *Priorities?*

- General purpose code
- Ease of programming
- Performance
- Portability



# *Background*

- Language choice
- Compiler choice
- Frame cycle time



*What's the secret to Cell programming?*



- \* Understand the architecture***
- \* Understand the data***
- \* Make the effort***



***Objection: “But I'm making a cross-platform game”***




***Most good solutions for the Cell  
will be good solutions on other  
platforms.***



***Where data and code optimization are merely important on conventional architectures, it's critical on the Cell.***



## *Typical module development -*

- \* Scalar C/C++ version*
  - \* PPE vector optimized*
  - \* Multi-threaded version*
  - \* SPE vector optimized*
  - \* SPE synchronization optimized*
- 


***“What modules should be put on the SPUs?”***



*All of them.*



***“WTF? Are you saying I need to re-write all my code to get performance on the PS3/Cell?”***



*High-performance code is easy to port to the Cell.*

*Poorly performing code from any platform is hard to port.*



- \* The Cell is not a magic bullet.*
- \* The Cell is not a radical change in high-performance design.*
- \* The Cell is fun to program for.*



# *The first skills to practice*

- \* SIMD optimization*
  - \* Lock-free synchronization*
  - \* Branch-free programming*
  - \* Dataflow design*
- 
-

*Oh, and...*

*Understand the restrict keyword  
and strict aliasing.*



***For some tips see:***

***[www.cellperformance.com](http://www.cellperformance.com)***



***“What's the easiest way to split programs into SPU modules?”***



***Let someone else do it.***



***But if you are interested in writing  
high-performance code yourself...***



*Number One Rule:*

***DATA IS MORE IMPORTANT  
THAN CODE.***



*In order to divide the application,  
consider:*

- \* Data locality*
  - \* Data generation patterns*
  - \* Data decomposition*
  - \* Data agglomeration*
- 
-

## ***Data locality: Streaming problem***

- Double or triple buffered***
- Load -> Store -> Update***



## ***Data locality: Sparse Data***

- ***Some kind of cache***

- ***Look at the access pattern:***

  - ***Structured access? Re-arrange the data.***



***Data locality: Bursty Graphing***  
***- Well-planned queues***

***i.e. Keep collecting the data***



***Data generation: ~ 1:1***

***- Divide and conquer***

***- Deal with edges separately***



***Data generation: ~ 1:many***

***- Synchronization will be the main issue, focus on reducing locks.***



***Data generation: ~ many:1***

***- This is the worst, often a sign of bad data design.***

***- Reduce the source data, focus on re-calculating.***



## *Data decomposition*

- Keep a dataflow graph*
- Each leaf may need a different solution*



***Data agglomeration:***

***- Look for imbalances in memory access and calculation.***

***Often faster to calculate than to access memory.***



***Basic Principles of Data Design:***

***Remember, it's all about the data!***



***Work closely with content  
creators.***



***Know the data and access patterns.***



***Be prepared to re-organize the data.***



*Everything must make a  
difference.*



*Design for the hardware.*



***Sort by dominant type.***



***Clearly distinguish read-only,  
write-only and read-write data.***



***Know that everything belongs to  
a set.***



*Take advantage of frame  
coherence.*



***What are you doing that's bad  
data design?***



***BAD: Domain-based object-oriented design.***



***BAD: Many layers of abstraction  
designed to hide code and data  
complexity.***



***BAD: Abstract interfaces and  
virtual functions.***



***Good code follows good data.***

***Fast code follows good data.***

***Small code follows good data.***



***“Do you really expect all the programmers to worry about how their data design affects performance?”***



***YES.***



*You can't optimize when it counts if you don't have the practice.*

*It's also useful for GPU coding.*



*Wrapping this up...*



