

PlayStation™ Optimisation



Hints and tips for improving
the speed of your
programs.



Allan Murphy. SCEE.



Introduction

- CPU Overview
- The gcc Compiler
- What the compiler generates
- I Cache Optimisation
- D Cache Optimisation
- Code Layout
- Miscellaneous



CPU Speed

- General
 - R3000A derivative
 - 33.8688 Mhz clock
 - 132 Mbyte/sec theoretical DMA speed
 - Cut down coprocessor 0
 - ie no MMU
 - Coprocessor 2 is GTE 3D chip
 - CPU speed is never optimal because....



CPU Speed (cont)

- Affectors: DMA
 - 7 DMA channels affect CPU speed
 - Deny access to RAM which stalls code
 - (unless you're lucky)
 - During DMA, CPU can only access:
 - internal registers
 - caches



CPU Speed (cont)

- Affectors: DMA sources
 - DMA to and from MDEC
 - DMA to and from GPU
 - DrawOTag - asynchronous GPU draw
 - CD to DRAM DMA
 - SPU RAM transfers
 - PIO transfers
 - OT clearing (ClearOTagR() only)



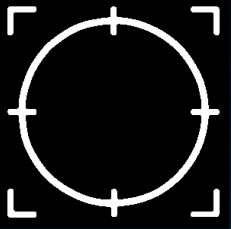
CPU Speed (cont)

- Affectors: Cache fill
 - Instruction cache automatically filled
 - Slot by slot transfer
 - Each slot is 4 32 bit words
 - 'Background level' of DMA
 - 'swings and roundabouts'



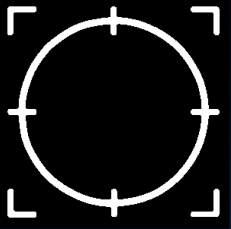
CPU Speed (cont)

- Affectors: Interrupts
 - Possible to seriously affect CPU speed
 - Examples:
 - CPU clock interrupt
 - Pixel clock interrupt
 - Heavy processing in callbacks



The compiler

- ccpsx
 - Written by SN Systems, part of Psy-Q
 - Triggers other compilation phases
 - Handles passing options to correct phases
 - Front end for gcc, aspsx and psylink
 - No potential for optimisation here



The compiler (cont)

- `cpppsx`
 - standard GCC preprocessor
 - Macros, includes, etc
 - Optimisation:
 - Make small functions into macros
 - Lose function call overhead
 - Possible 'code bloat'
 - No serious effect on readability



The compiler (cont)

- cc1psx
 - GCC object code compiler
 - (w/PSX modifications)
 - Handles C -> assembler conversion
 - Most scope for optimisation
 - See later
 - (cc1plpsx is C++ version of cc1psx)



The compiler (cont)

- aspsx
 - SN's assembler
 - Used by compiler only
 - Not a macro assembler (eg asmpsx)
 - Not responsible for optimisation



The compiler (cont)

- psylink
 - SN Systems object linker
 - Links your code & objects with libraries
 - Builds final executable
 - Responsible for code positioning
 - Use map file for optimisation
 - Not responsible for any optimisation



The compiler (cont)

- dmpsx
 - Postprocessor written by SCEI GTE team
 - Converts GTE macros inside program text
 - Builds real GTE cop2 instruction sequences
 - Allows interleaving of CPU tasks with GTE tasks
 - See GTE presentation for more details



Compiler Options

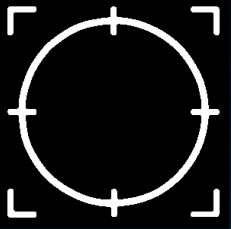
- -g
 - Forces no optimisation
 - Does not re-order instructions
 - Does not attempt to remove delay slots
 - Allows debugger to step through C
 - Fixed expressions replaced with their value
 - No variables in registers (except parameters)



Compiler Options (cont)

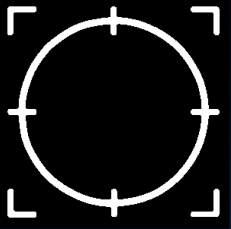
➤ -O1

- First level of optimisation
- Local variables put into registers
 - (Compiler's decision which variables and which registers are assigned)
- Delay slots filled
- Repeated expressions removed
- Unneeded locals removed



Compiler Options (cont)

- -O2
 - As per O1
 - Turns on all but 2 optimisations
 - Eg Frame pointer elimination
 - More clever register allocation
 - Some make no difference in R3000



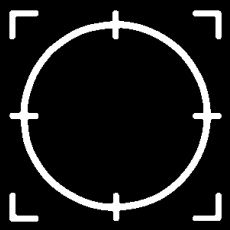
Compiler Options (cont)

- -O3
 - As Per -O2
 - Also compiler inlines functions
 - Heuristically chosen
 - Compiler unrolls loops
 - When loop iterations known
 - gcc supports more obscure optimisations
 - Not certain to provide improvement



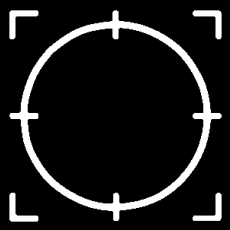
What the compiler generates

- General R3000 Info
 - RISC processor
 - only 1 load instruction
 - only 1 store instruction
 - Instructions all 32 bit
 - Synthetic instructions (macros)
 - 32 bit addresses have to be 'built'
 - Some instructions require a delay slot



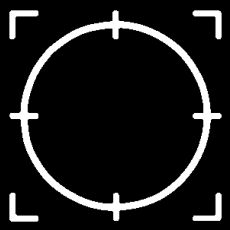
What the compiler generates (cont)

- Register set
 - 32 general purpose registers
 - Orthogonal design
 - 2 registers for division results (HI / LO)
 - Interrupt / exception registers in Copro 0
 - Hardwired zero



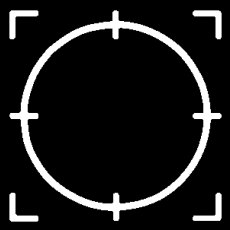
What the compiler generates (cont)

- Compiler conventions
 - 1 assembler temporary
 - 2 for function returns
 - 4 for parameters
 - 8 saved locals, 10 locals
 - stack, frame, global data pointers
 - return address
 - Interrupt handling



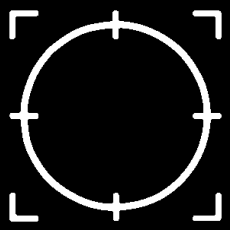
What the compiler generates (cont)

- Assembler usage with C
 - Parameters, return values, temporaries and assembler temp all fair game
 - 17 registers available
 - Careful with gp, sp, fp and ra
 - Don't touch k0, k1
 - Use s0-s7, but save and restore
 - In raw assembler, no conventions



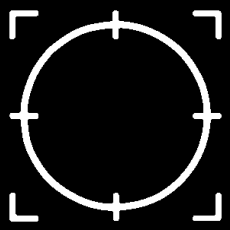
What the compiler generates (cont)

- Reading & writing data
 - All addressing 16 bit offset to register
 - Full 32 bit addresses built in 2 stages
 - Followed by or combined with load / store
 - Locals faster than globals
 - Read / write FIFO
 - DRAM Write up to 5 cycles
 - D-cache write or read only 1 cycle



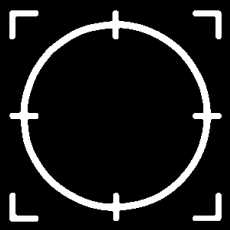
What the compiler generates (cont)

- Marking instruction sequences
 - Via `__asm__` (“.....”); construct
 - Output recognisable sequence
 - Examine disassembly
 - Code sequences marked
 - Possible to emit assembler directly in C
 - Careful with register convention



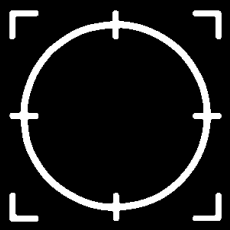
What the compiler generates (cont)

- Function Parameters
 - First 4 parameters passed in registers
 - A0-A3 (compiler convention)
 - Assuming 4 byte or smaller parameters
 - Thus including pointers
 - Functions with > 4 params are slower
 - Since extra params are pushed onto stack



What the compiler generates (cont)

- Locals
 - On stack with -g
 - In temporary registers with -O or 'register'
 - Compiler chooses
 - Temporaries saved across function call in saved temporary registers
 - Functions with 10 locals or less faster
 - Load/store offset from fp (faster)



What the compiler generates (cont)

- Globals
 - Full 32 bit address must be calculated
 - Since globals can be anywhere
 - Exception - when in sdata / sbss section
 - Loads/stores offset from gp
 - But only 64Kbytes worth of data
 - Large data structures slow to access



I Cache Optimisation

- On Chip SRAM instruction cache
 - Automatically filled
 - 4Kbytes
 - direct mapped
 - Slots are 4 words
 - Big impact for cache misses



I Cache Optimisation (cont)

- Code Layout
 - Avoid cache misses
 - Lay code out carefully
 - Consecutive 4K chunks map to same area
 - Regular jumps in code cause thrashing
 - 4K alignment does *not* help speed
 - Move common functions closer



I Cache Optimisation (cont)

- Code Layout
 - Stay inside 4K bytes for critical routines
 - I.e the core processing functions
 - -> implies writing in R3000
 - Use map file to detect overlapping routines
 - Bottom 12 bits of address determine cache block



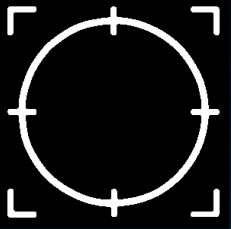
I Cache Optimisation (cont)

- Loop Layout
 - Use small loops
 - Minimise regularity of jumps
 - -> Stops cache refill happening so often
 - Large loops miss cache every iteration
 - Thus much slower



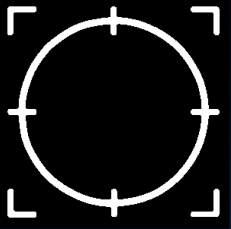
D Cache

- On chip SRAM data cache
 - Under programmer control
 - Not filled automatically
 - 1Kbytes
 - 1 cycle read / write
 - Base address 0x1f800000
 - Top address 0x1f8003ff



D Cache Optimisation

- Stack on D Cache
 - Set stack pointer to top of D cache
 - Stack grows down through cache
 - Do not use more than 1Kbytes stack
 - All locals on cache, 1 cycle read / write
 - Speedup around 10-15 %
 - Dependent on data organisation and usage



D Cache Optimisation (cont)

- Variables on D Cache and sections
 - Directly declare variables on the cache
 - Use compiler pragma
 - Forces section for variable
 - Faster than making a pointer
 - (since pointer address must be loaded)
 - Must copy initialisation data to cache



D Cache Optimisation (cont)

- Declaring D Cache Variables
 - Best done with `#define`
 - Must initialise the variable
 - Or else section pragma ignored
 - Cannot use pragma for locals
 - For locals, use stack on cache
 - Pragma can be used to force code section
 - See example assembler / link file



D Cache Optimisation (cont)

- Declaring D Cache Variables
 - Initialisation values in named group
 - Group in main RAM
 - Copy from DRAM to D cache
 - Get size from grouporg and groupend



Code Layout

- Map files
 - Generated by psylink (/m)
 - Shows location on code
 - non-static functions
 - non-static global variables
 - locations of groups / sections
 - Use to check I cache conflicts
 - 'Missing' RAM



Code Layout (cont)

- Sections: text, data, bss, heap, stack
 - Text contains executable code
 - Data contains initialised variables
 - Bss contains uninitialised variables
 - Stack is space for locals
 - Heap is the rest of DRAM remaining
 - CPE/EXE contains text & data only



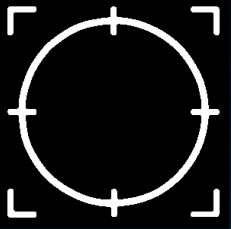
Code Layout (cont)

- Sdata, Sbss
 - Special 'short data' sections
 - For direct access from the gp register
 - Variables stored within 16 bit offset of gp
 - gp fixed at startup time
 - And fixed throughout
 - Direct load / store
 - No 32 bit address building



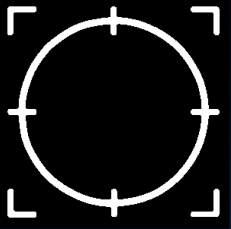
Code Layout (cont)

- -mgpopt, -G<num> & gp
 - Force compiler to put variables in sdata / sbss
 - -mgpopt forces gp optimisation
 - -G to specify maximum size in bytes
 - Cannot have more than 64K bytes of data in sdata / sbss combined
 - Link error will occur in this case



Miscellaneous

- inline
 - Code for simple functions inserted in caller
 - Function call overhead removed
 - Not across object modules (cc1psx, linker)
 - Not with -g
 - -O1 and above may inline automatically
 - Compiler chooses functions to inline
 - Or force with 'inline' keyword



Miscellaneous (cont)

- register keyword in C
 - Only has any effect with -g
 - With -O<num> compiler makes choice
 - Compiler puts variables in registers
 - Possible to force a variable into a register
 - But not recommended
 - Compiler may make a better job of register assignments than a programmer



Miscellaneous (cont)

- register declarations
 - Example:

```
register int fastVar asm( "$8" );
```
 - fastVar now stored in register 8 (t0)
 - Compiler cannot use registers you assign this way in the variable's scope
 - Cannot take address of register variables
 - Take care with compiler's usual assignments