A Crash Course in SESC

This document provides you with essential information in getting started in using, understanding, and hacking SESC. Please read closely and let Andrei know of any hitch you may encounter on the way to SESC Nirvana.

Installing SESC

1. Copy recursively the directory
   /cse/courses/cse471/08sp/sesc

into a directory of choice (e.g. your home directory). I will refer to that directory as **/path/to/sesc** henceforth.

2. Go to that directory and read the README file, e.g. **less README**. The next steps are loosely inferred from that file.
3. **mkdir ~/sescinstall**
4. **cd ~/sescinstall**
5. **/path/to/sesc/configure --enable-smp --enable-debug-silent**
6. **make**
7. Don't forget to pass --enable-smp to configure – we need the SMP build.
8. Use sescutils straight from **/cse/courses/cse471/08sp/sescutils**. In theory you could build them too, but (1) it takes until the cows come home, (2) there are tweaks you need to do to get things done. The sescutils toolkit provides:
   - Cross-compiler
   - Disassembler
   - Debugger
   - Other similar utilities

It is unlikely you'll actually need sescutils, as you are already provided the usual SPLASH-2 suspects in binary form. We recommend you use the **fft** and **lu** programs. They don't take a very long time to simulate, yet are complex and interesting enough to exercise your cache protocol fully.

WARNING: Building SESC on the department machines required altering several configuration items on both the department system and the SESC source code - some helped by the coauthor of SESC herself! Unfortunately at this time you cannot install and run properly SESC on a local machine, e.g. your beloved laptop or home desktop (I tried real hard...) You may want to avoid the current SESC distribution online as it is incompatible with our setup. Also, please avoid like the plague ALL files in the directory **/path/to/sesc/src/libtm**. Currently that dir is needed for building, but we have severed all ties with it so you shouldn't worry. Just don't touch it – it has significant bugs.

HOT TIP: Get SESC built early and understand how to rebuild it any time you want. You don't want to fumble with building when you need to focus on higher-level activities.

Ten Steps to Understanding SESC

Here are some tips for understanding the basic architecture of SESC. Some of the information I gave you in class was outdated. In the typical manner, the documentation hasn't kept up with the code (e.g., “We haven't used **InterConnection** in five years!”) There is nothing like a face-to-face meeting with the author.

1. Take a look over the dox at http://iacoma.cs.uiuc.edu/~paulsack/sescdoc/. Read the overview but don't get bogged into details. Also see http://cs.washington.edu/education/courses/471/08sp/hw/sesc-slides.pdf.
2. Look in **src/libcore/*** to get an overview of the functionality.
3. You will make extensive use of configuration files. Take a look at files in **confs/*.conf**, particularly **cmp.conf** and **smp.conf**.
4. You will use a simple bus architecture. Take a look at the files **src/libmem/Bus.*** and **src/libsmp/SMPSystemBus.***. They document what primitives the bus offers. WARNING – the bus is not entirely synchronous: the operation of sending a memory request and getting it resolved is not atomic. However, the collection of responses to a request from all caches is atomic. The bus holds a container of all Cache objects in the system, and each Cache object has a back-pointer to the bus. They are called “upper level” and “lower level” pointers in SESC terminology, you will see them in the source.
5. See **FetchEngine.*** for the fundamentals of fetching and decoding instructions. See **GProcessor.*** for information on the processor (fetch width, issue width, queues etc.) The CMP and SMP architectures described by the homonym files in **confs/** (and mentioned above) describe systems with several processors, each with a cache subsystem connected to a common bus.
6. The memory requests are documented in **MemoryRequest** and **DMemoryRequest** classes. Contrary to what I said today in class, “D” stands for “Data”, not “Dynamic”. There is an **IMemoryRequest** that is an instruction request and is not of much interest to you. You only care about the traffic of **DMemoryRequest** objects.
7. Memory requests travel from cache to the bus to the other caches and get filled with relevant info. Your mission, should you accept it, includes figuring out what steps you need to take to orchestrate the proper exchange of such objects.
8. We recommend you use a simple write-through L1 cache and no L3 cache. Your focus will be around the L2 cache.
9. Now that you have an idea of the basic architecture, let's focus on the cache. Look at **src/libmem/Cache.*** (base class for all cache coherency mechanisms) and its derivees **src/libsmp/SMPCache.***. Do not look at **src/libtm/TMCache.cpp**.
10. The library comes with an implementation of the MTI coherency protocol. This is of great help to get you started with implementing your own. MTI uses three states and is a great example of what steps you need to take to hook your code into the simulator. Take a look at **src/lbsmp/MSIProtocol.***.

Let Andrei know if you have any questions! Good luck!