Now that I promised that the Propeller II array technical specification is to be final – I am going to emphasize a bit about the seed computer I am still planning to be building. I will cover the specification in a bit.

However, the requirement for the seed computer isn't going to be any different from the first report I wrote back in the Summer: It has to do few tasks – 1. Download the SPIN files or even just the uncompiled C++ text files. 2. Analyze what the program wants (I may put in the SPIN interpreter into the main OS in the seed computer to find out what's the objectives). 3. After analysis, set up the RAM buffering before transmitting the data onto the dispatched FPGA's DDR DRAM chips which the Propeller II chips will be given permission to peek into. 4. Download the buffered and potentially recompiled Propeller-II compatible ASM binary files onto the FPGA's RAM memory then dispatch the Propeller II chips by calling for the firmware ROM ID and the nested interruption vector stacks. 5. After being called, the Propeller II chips download the data onto its own SDRAM (or DDR) chip and then commence the tasks. 6. After it's over and the Propeller II chips are satisfied, they will send the data back to the seed computer via the FPGA chips. 7. Given the choices, the seed computer can either: Send the results back to the person who submitted the software to get it done via Ethernet (and afterward, the Internet), save it onto the hard drive, burn it onto the CD or DVD, and / or write onto the flash media (or another hard drive) via the USB 2.0 ports.

Now that it's covered, it's time for the specification (been thinking about it):
Western Digital 1 Terabyte Caviar Black SATA II 3.0Gb/s hard drive
4GB (x2 2GB) dual-channel DDR-II DRAM SODIMM laptop memory
AMD (formerly ATI) Radeon HD 4350 (or 4670) PCIe video card
Blu-ray Disc SATA burner (for to back up some data onto the BD-RE when needed)
All-in-one flash memory drive with floppy disk drive built-in
4U server chassis with 750W ATX switchmode power supply (LOT of interior room!)
60mm copper heatpipe (to be applied with Arctic Silver 5 thermal paste)

All of the parts are fairly standards. It's very easy to replace or upgrade. However, note the 60mm fan size on the heatpipe, can you guess what CPU I am going to use? That's right – Freescale MPC8377 PowerQUICC II pro network communication microcontroller (along with PowerPC G2 – e300 CPU core already built-in). Why? It's easy to be able to depend on something already future-proofed (even although it's a bit older by now), that is, the MPC8377 already have PCI-express bus (for the video card so I can see what's going on), USB 2.0 UHCI host controller (for the keyboard and mouse – and the additional mass storage hardwares), and finally, the SATA AHCI host controllers (two SATA ports – for the BD burner and the hard drive). And, it's pricetag isn't too bad: Starting at $40 for the 400MHz speed grade, even though I may want to be using 800MHz one going for $60 to $80 a pop. Why need faster one? I will explain.

As far as I am concerned, the SPIN interpreters are rather fussy – they painstakingly put two and two together, as far as the Propeller microcontrollers are concerned (even on the Propeller II chips!!!), I felt that it's best to up the PowerPC CPU core speed to 800 Megahertzs speed so the PowerPC processor can execute the SPIN and then perform the surgery on it then change the codes if needed (to prevent the electrical rule violation whereas the array of Propeller II chips are soldered onto the separate boards, that way the chips won't give up its precious magic smokes. The “wrong” pins usage will be removed during the analysis stage), then recompile it into pure ASM binary codes – all of the tasks without wasting the time.

“But, what about the video?” The remaining unused Propeller II chips will be called into the task of interpreting the video PLL data, and then write the PowerPC CPU's video vectors (on the main RAM) before it's handed off to the video card – only it will remain the same as what's intentionally programmed (kinda in a way the GEAR Propeller emulator works, only faster) – the video PLL waveforms won't be present anywhere on the Hypercube array board because the signal crosstalk contention rules have to be followed all the times. What comes out of the Propeller II chips will be spread-spectrum signaling data (kinda like the radio-transmitted white noises, only it's transmitted down the traces, carrying the data at extremely high speed), so the video PLL data will be passed to the vacant Propeller II chips to be transformed into the VESA data to be put into the hand of a PowerPC CPU.

Why PowerPC? The microcontrollers based on IBM's POWER architecture are pretty sophisticated, and in some case, cheaper than buying the DSP while it can be programmed to do exactly the same mathematic instructions execution (very few PowerPC MCUs do have AltiVec FPU already built-in). I see so many PowerPC microcontrollers leaving the DSPs in the Twilight already. What an expensive $140 single core DSP has and can do, an MPC555x microcontroller got it, at fraction of the chip price. Hence, that's the reason I picked the PowerQUICC processor for the seed computer as it's also already popular in Linux hacking community, next to the 64-bit multicore x86 microprocessors (few are of PowerPC variety and AMD's PowerPC-based x86 CPUs are quite popular with PC gaming too).

I really wanted to use AMD Sempron in my seed computer, but the issue is, the HyperTransport IP for the FPGA is quite expensive (and I doubt they're open-sourced – I have heard that HyperTransport Consortium have already made few HTT IP open-source, but I am not positive on that), so I decided I should settle onto the PowerPC microcontroller (as I can just make an x86 interpreter and put it in the firmware ROM, anyways).

“So, what about the bootsector? The firmware?” I have two options, U-Boot (Linux boot firmware that the Coreboot's based loosely on), or the Coreboot itself. uCLinux is one more option, but I had to put it in post-boot time as I have heard that it's kinda troublesome to bootstrap the uCLinux for the PowerPC (and that Freescale gave up on uCLinux so they decided to let the mortals try – very few succeed, so it should be in uCLinux toolchain trees by now, however, I prefer to use what's already well-known to work). It's also better than compiling something entirely on my own and then take a shot in the dark. Linux's 100% configurable, so I don't really have much to lose, though.

About one terabyte hard drive, I have another problem. “What is it?” You see, many 1TB hard drives requires 4KB sector clustering (NTFS volume mounter in Windows automatically do that when it sees a blank storage volume way too big for the 512 bytes sector “labeling”), I am wondering about the reliability of SaneFS (Ext2FS – an official Linux volume), so I may try using the partition method I used in my own homebuilt OS I built a long time ago – Advanced Technology File System (ATFS) with 4,096 bit RHC (Random Hash Clustering) encryption. But since I may be allowing some public usage for the requesting users, so I am not going to encrypt the whole volume, instead keep the encryption key table for the 256-bit (or 512-bit) CRC checksum keys for the defragmenter (when it really need the housekeeping). If I chose to use 512 bytes clustering option, the hard drive's VLIW DSP chip will get, shall I say, quite confused. Also, I elected to use the Serial ATA (SATA) interface as the IDE hard drives are quickly disappearing into the long, forgotten corner of computing history, like that lonely MITS Altair 8800 computer sitting in your closet being neglected. (Of course, if you use Altair 8800 every day for the pleasant trips down the memory lane from the middle 70s, then that thing's lucky.)

And, the SATA hard drive have one software-based technical specification that ATFS volume mounter can actually take advantage of when it need to retrieve the data from, write onto it, and / or defragment it pretty quickly (after all, 1TB's a huge space), it has the NCQ (Native Command Queuing – that allows the CPU on the hard drive board to skip the certain area of the platter, and can also do out-of-order head positioning, allowing insanely fast data readout) that the OS can have the hard drive running its data in warp speed.

Okay, about the video card. Why need one? The answer: It's very, very easy to use as the VESA programming manual's open-source already (there are some open-source GPU drivers we can use too but I will also need to use DRI, a generic supercomputing video card driver for AMD Radeon HD families), and they're pretty common. What's also great about it, is the HDMI connectivity – I can actually plug in the HDTV (or the projector) and then demonstrate the Propeller II array's prowess, in the front of Expo folks. And, it's straightforward to upgrade or replace whenever needed.

What you see as of the PC hardware, it's standard already so it makes sense to depend on it – if that part dies, no problem-o. Slap in new one, and it's up and running again. However, understanding the violent bandwidth requirements, I decided to use the reputable hard drive brand, Western Digital. Western Digital Caviar Black hard drive is designed with reputability and reliability in minds, so it's best to use it. Caviar Black is also used in servers and TV set-top boxes. So, that do makes sense in using it. Why 1TB? Supercomputers are hard drive hogs. I have to set up the margins for myself and the CPU arrays, so it won't fall apart as it's churning out mad numbers. I wouldn't worry too much about the huge garbage stacks (mostly memory swaps) as it can be safely eradicated after the jobs are finished (the same way Windows and Linux deal with memory swaps – that includes MacOS too).

As far as the standard PC parts goes, they first are expensive, but get cheap quickly, I figured I should take advantage of easy-to-obtain parts together in this hardware, with a cheap but extremely powerful microcontroller, Freescale MPC8377.

The case? I am thinking about getting the server case with less of the expensive frills. Removable hard drive tote? Maybe, but I doubt both Windows and Linux would ever understand the ATFS partition volume – not without tons of works, and someone wouldn't be happy about installing non-standard volume mounter into their OS workspace in their servers. I picked 4U rackmount as it got BIG free spaces (WASTE of space for some Average Joes, but me? I am a happy camper as I can put two different computers altogether in one case). What's left of the PCI slots aren't going to matter anymore, I can cram all the supercomputer boards in the otherwise wasted area of the server case, while other half area of the server space have to be wide open as the MPC8377 needs lot of cooling. (Remember, I picked 800MHz speed grade so it can bulldoze through the SPIN execution during the analysis stage and then hand over recompiled ASM codes to the Propeller II chips, within the limited times.) Also, the rackmount is quite nice as the whole affair is on four pivotable wheels – a back saver.

And, the seed computer can be quite efficient too (the 800MHz PowerPC MPC8377 consumes 250mA at 1.1 Volts DC), as well as the Hypercube array (Propeller II eats 270mA at 1.8V, not as efficient as a main CPU, but at least it can be programmed to throttle its core frequency and shut down unneeded Cogs to save extra few watts). I would use MPC8640D if I wanted to, but it's not as efficient as its brother MPC8377 (although six times more powerful than MPC8377), nor do it have the SATA IO options. And, while MPC8377's already cheap, MPC8640D is not, so I would just save it for something more fancier and harsh. After all, this supercomputer's a hobbyist DIY project (also an expensive one at that).

And, yes, I have MPC8640D processor – that thing's neat! But, the price is so prohibitive that it's best to “hire” another kind of CPU (of the same ISA architecture, PowerPC) to really reduce the overall hardware pricetag. MPC8377 is better suited for this job.

I am thinking about using FRAM (or MRAM – if there is a SRAM port on MPC8377, no problem, I can hack it to boot up through that port – to the MRAM containing the firmware's OS kernel). It's to allow the field upgradability without imposing the wears on the firmware RAM.

I am also thinking about using military temperature SiTime MEMS oscillator (unsure if I can do away with SiTime Encore TCMO), which would be the same as the ones soldered on Propeller II array boards. The temperature endurance threshold would offer this computer wide margin so its clock won't be dancing, or worse, falters. Just a falter on the clock pin can fry a CPU – have seen it happening on the overclocked computers. I wouldn't care about about 2% Integer Unit core frequency hopping (it's unavoidable in some case, but I will stuff the CPU area of the motherboard with lot of decent low-ESR decoupling capacitor to ensure it won't happen within the PLL circuitry – a requirement for dealing with a DDR II memory infrastructure – they have very strict clocking requirements).

Why would I use MEMS oscillator? I like its features, especially its unbreakability (they were tested up to 30,000 Gs, and the tiny moving parts composed of Silicon are intact throughout the punishing tests, which the Quartz oscillators wouldn't survive), its superior programmability while retaining jitter PPM performance (compared to the programmable Quartz PLL oscillators – some of them were horrible and the RC oscillators were way worse due to its temperature sensitivity. Propeller have RC oscillator built-in – by RCSLOW and RCFAST, and they are notoriously worse than both MEMS and Quartz oscillators but is fine for booting up, though.) Programmable MEMS oscillators are quite cheap, so I figured I would use every favors for the usage in the basis of clocking circuitry as they are also pretty efficient (3mA current consumption).

Even though I am going 100% with the MEMS oscillators in my own supercomputer, I can't force you to abandon the Quartz oscillators. Actually, MEMS or Quartz, it doesn't matter as long as they're rated appropriately for the listed duty cycles and ripples (another word for the clock jitters). What sets them apart are of few features: Survivability. Clock jitter toleration. Programmability. Power efficiency. Clock stability. That's more of your decision in choosing the clock generator in your design. For me, long-term (and even short-term) clock stability's very important in this computer.

With few things put asides, I am going to answer very few questions. “How can the jittery clocks fry a CPU?” Simply put, the CPU's core clock PLLs depends on squeaky clean clock pulses on which it can lock its heterodyned clocks onto, to be able to make copies of known clean clock pulses and multiplies them into extremely high frequency clock pulses which the CPU circuitry depends on as the reference clocks. When the clock drops out or becomes jittery, the PLL loses lock onto the clock, or trying to lock onto what's now a dead clock pin (PLL latches on at the rising edge of clock pulse, while some latches on falling edge of the clock pulses), the PLL tries to make up for the loss of lock, by oscillating – which in few case, like with horrible DC power rail filtering, it can break into the chip-killing rogue oscillation, which still mean nothing to the CPU circuits – therefore, they all short-circuit. Short-circuited CPU is a dead CPU. With some CPUs, the PLLs comes with the fault-tolerate designs, which if the oscillator drops dead, the CPU shuts down instantly without any damages being done. It really depends on the wiring configuration (such as Brown-out Enable tied to Vdd), another reason to check the datasheet before designing the printed-circuit boards and/or soldering the chips.
I have, however, heard of the Propeller microcontrollers getting fried by the jittery or faltered clock.

That is, the reason why the clean clock is so important! Okay, onto the next question. “Why use CD? We already have the SD memory card.” I can use BOTH. Actually, the CD-RW (or BE-RE) option is just to ensure the data isn't going to disappear tomorrow (data loss disaster can happen anytime, much to my dismay). I also have chosen to use a hard drive, so I can just hold the finished result today and then retrieve it tomorrow or next week onto the SD card. “Okay, I got it. Why not make a hard drive removable? Like in a server hard drive tote?” I have decided that there's no point because of the non-standard partition, which makes the job of the others and the OSes on their computers harder. I would use the standard partition if I wanted to, if not for the 4KB clustering, though. I will have to look into few partitioning options before considering making it removable (remember, it have the master OS on it too, that is, if I unlock it and remove the hard drive, and attempt to boot the seed computer up, it will ask for the bootable volume which is no longer present – now in my hand). “Okay. Okay, I see. Why Linux?” Linux's designed from the very beginning (1990 – '92), to be totally open-sourced and configurable, that way the projects (even some of expensive kid toys!) can be programmed to the way its creator wanted without having to create the complex infrastructure of the OS (been there, done it), instead the GPL (Gnu Public Licensing) is fully realized just FOR you to do what pleases you, such as studying how the kernel operates (Linux is especially interesting), or how the “calls” are put up on the kernel's hooks. “So, you mean U-boot is linux?” Yep. It was originally created for PowerPC microprocessor (very few hacked Macintosh uses it), now that it's starting to infiltrate the x86 area of computing. Open Firmware is neat, but actually, Linux is much better in term of crash handling and other unforeseen events which just make the Award BIOS and the AMI Megatrends BIOS spit out the hairballs. Award BIOS is a bit better in most of the respective points of booting computers than few of the BIOS. Coreboot is 'nother of Linux-based boot firmware. And a nice one at that.

I had to be a bit careful with “what I really wanted”, unlike Homer Simpsons who picked way too much of the greatest things to be put into a station wagon at General Motor. Eeek.

Just jokin'. ^____^

But still, having great features wouldn't hurt, though – that could save me much headaches in the roads. Sure, I will have big headaches now as I work on it, but I will be thanking myself in few years when I get to use that features while I get on that problems in the SPIN files. One is SPIN analysis and execution (which the PowerPC processor tests for any funny behaviors and remove the bugs, and keep what's not a bug but a feature and that lists will just keep expanding due to the “unexplored sea charts” in the new Propeller II chips' programming methods) – which I will be glad that I am going to put into the firmware ROM, saving me the possibility of smoked Propeller II chips.

So, if you ask, “PowerPC processor runs SPIN? Why?” - it's mostly for the protection: Way too much smoked Propeller II chips will simply means EXPENSIVE REPAIR. So I would like to be able to avoid that problems in the first place. PowerPC processor just execute it for analysis and hand it off to the Propeller II chips via the crossbar switch (FPGA), and that's it. That processor won't have to keep whirling on it for too long (unless the SPIN binary file's a monster, which is just impossible largely because of the memory limitation – the Propeller IDE just stick to PMM [Propeller Memory Model], which is now both 32KB and 128KB).

Therefore, that's the progression I am trying to work painstakingly on to get it to be the success I am looking for. And, Propeller II's a very nice hardware, so I have decided to start Dendou Oni (Electric Demon) supercomputing project, to prove the Propeller II's prowess, and be quite useful at the same time. And, finally, I would like to thank Chip Gracey and his company, Parallax, Inc.!