

“This thing we call failure is not the falling down, but the staying down.”

— Mary Pickford

AUTOMATED APPLICATION LEVEL CHECKPOINT-RESTART

WHEN BAD THINGS HAPPEN TO GOOD PROGRAMS

Living in the information age means, among other things, being aware that computers fail. Disks crash, viruses invade, bad things happen. With PCs, prudence dictates regular backups. If you're a scientist running simulations on a supercomputing system, which may take days, weeks or months to complete, prudence dictates checkpoint-restart — CPR for short.

CPR means that your “code” is designed and equipped with checkpoints, places where during execution the program periodically saves the results up to that point, so that if a crash occurs, you can restart from the last checkpoint with minimal loss. The concept is basic, if not always faithfully practiced.

To facilitate CPR on LeMieux, PSC's 3,000 processor terascale system, PSC senior research analyst Nathan Stone developed a very fast set of I/O routines — called TCS-CPR — to save checkpoint data to disk, back it up, keep track of it and automatically restart a program that crashes. “Automated restart,” says Stone, “is a big sell for researchers who like to sleep.”

TCS-CPR neatly takes care of everything associated with CPR except the fallible human part of manually writing checkpoints into the application code. This can be an onerous task and for some complex codes there's not necessarily an obvious place to insert checkpoints. At the same time, with the growing prevalence of large-scale parallel systems like LeMieux and the large-scale, long-running simulations they enable and, even more, the movement toward grid computing, the risks of failure

are increasing, trends that make CPR more important than ever.

Recognizing this, Cornell University computer scientist Keshav Pingali and his team devised a scheme to save human time by automating the code-writing part of CPR. With their approach, called C³ (Cornell Checkpoint pre-Compiler), a pre-compiler program takes uncheckpointed source code and transforms it to become self-checkpointing and self-restartable. In collaboration with Stone and other PSC staff, they have tested C³ on several platforms including LeMieux and a Windows cluster at Cornell. Their results show that C³ introduces little overhead into program execution and demonstrate the feasibility of their innovative approach.

Keshav Pingali,
Cornell University and
Nathan Stone, PSC



“MANY PEOPLE WOULDN'T BELIEVE THIS IS POSSIBLE, BUT THEY'VE DEMONSTRATED IT.”

The Goal: Full CPR Support for Big Ben

CPR took on heightened importance in 2001 at PSC with the arrival of LeMieux, the National Science Foundation's first terascale system. Comprised of 3,000 processors, LeMieux provided NSF researchers with unprecedented capability to carry out very large-scale simulations. It also meant, for simulations that used hundreds or thousands of processors simultaneously, a statistically higher likelihood that something bad would happen on one of those processors during execution.

“Until recently,” says Pingali, “most parallel processing was done on relatively reliable big-iron machines whose mean time between failures was much longer than the execution time of most programs. But many programs now are designed to run for days or months on even the fastest computers, even as the growing size and complexity of parallel systems makes them more prone to hardware failure.”

Another pressing reason to pay increased attention to CPR is grid computing. As cyberinfrastructure such as the NSF TeraGrid gains maturity, it will become more common to run applications on geographically distributed systems, with different systems teaming on different parts of the job. “With grid computing,” says Pingali, “the probability of failure becomes much higher.”

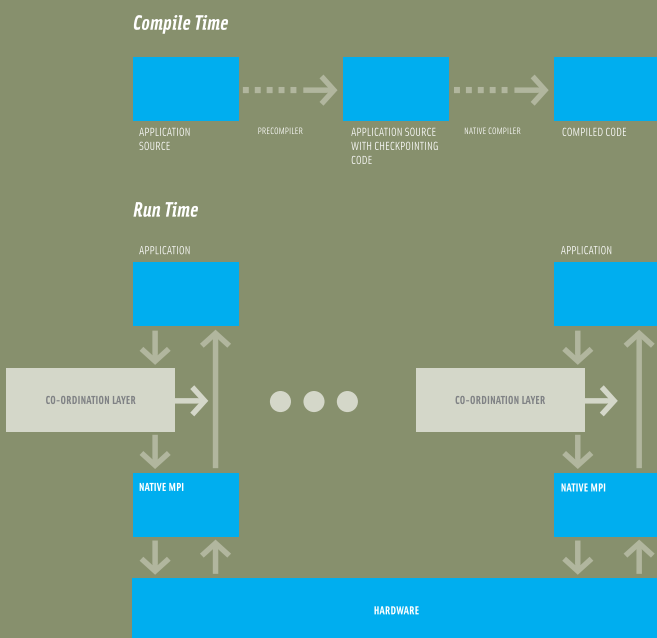
With TCS-CPR, Stone's idea was to provide robustness and reliability in the checkpoint write and restart operations. Simply by replacing the open, write, close statements in their source code CPR routines with TCS statements, users gained a fast connection to disk, the reliability of checkpoint backup — by virtue of an offline process that replicates the checkpoint data and keeps track of it — and automated restart in the event of a crash.

Despite the advantages, users did not rush to implement TCS-CPR. Faced with this situation, Stone a few years ago became involved in a CPR working group at Global Grid Forum. There he met Paul Stodghill of Pingali's group at Cornell and realized their work complemented his with LeMieux. “They knew smart ways to automate writing CPR into code,” says Stone, “and we knew how to save and keep the checkpoint data in a robust way and give it back to the application so that the user has no idea it was ever interrupted.”

In the collaboration that ensued, Pingali's group used LeMieux as a test platform for C³. They tested different application codes to evaluate the amount of overhead — added execution time — involved in bookkeeping code that C³ inserts to implement CPR. On all among a large range of codes, this overhead is less than 10 percent, and in most cases around 2 to 3 percent. Further testing evaluated the overhead cost involved with saving checkpoints during execution. With checkpoints done hourly, this overhead is under 4 percent — a small cost for the security against failure gained from CPR.

A further objective of Pingali's CPR scheme is portability — being able to restart applications on a different platform than the one they start on, a facility that envisions grid computing. In one experiment, they applied their approach to this difficult problem — implemented with software they call PC³ — by taking checkpoint files from an application running on LeMieux and restarting it on a Windows cluster at Cornell. The job ran to completion on the Windows cluster with identical results to the LeMieux process. “Nobody's ever done this before,” says Stone. “Many people wouldn't believe this is possible, but they've demonstrated it.”

Stone and others at PSC are now collaborating with Pingali's group to test and implement other checkpoint techniques on Big Ben, PSC's 10 teraflop Cray XT3 system and a lead resource on the TeraGrid for the most demanding large-scale parallel applications. “Our goal,” says Stone, “is full production-level CPR support for the XT3. We're going for the five nines, 99.999 percent reliable.”



Overview of Cornell Checkpoint preCompiler

C³ takes source codes in C (programming language) or Fortran with MPI (message-passing interface) and instruments them to do application-level CPR. The only requirement for programmers is to insert a simple statement — a “pragma” — in the source code to identify points where checkpoints might be taken. The C³ run-time system determines whether to write checkpoints at those locations.

The precompiler output is compiled with the user's native compiler of choice for the hardware platform and linked with a run-time library that constitutes a “co-ordination layer,” which sits between the application and MPI library and intercepts all calls from the application program to the MPI library. The coordination layer is designed in such a way that it is unnecessary to modify the underlying MPI library.