User and kernel level checkpointing

PROGRESS project

Norbert Meyer

Introduction

- Checkpoint on different levels:
  - application level
  - user level
    - library
  - kernel level
    - kernel modifications
    - kernel module

Development

- User level
  - psncLibCkpt library
- System kernel level
  - psncC/R package (kernel module)
- Testbed environment
  - Sun Blade 100 (1 processor)
  - Sun Fire 6800 (SMP architecture)

psncLibCkpt library

psncLibCkpt

- The psncLibCkpt is based on libckpt library which allows making „basic" images, further developed and enhanced by PSNC
- This is a library for applications written in C
  - few changes are required
    - modification of main() function in application code
    - application has to be compiled with additional compiler parameters/switches „-lrt", „-ldl"

Functionality (1/3)

- Supports multiprocess programs in „one parent – many children" model
  - parent process cannot be grandfather
  - child process may use exec() function
  - mechanism for correct handling of wait() function
- Saves and restores memory segments used by shared libraries

Functionality (2/3)

- Supports IPC mechanism
  - System V semaphores
  - System V shared memory
  - System V message queues
- Introduces virtual keys and identifiers of IPC resources

Functionality (3/3)

- Saves and restores signals disposition
- Saves and restores
  - pending signals
  - signal mask
- Extended support for open files
  - restores content of opened files

Implementation requirements

- Multiprocess applications has to follow „one parent – many children" model
- All child processes need to finish their computations before parent process
- Only one change in source code required:
  - replace main() function name with ckpt_target() name
- Some system calls are forbidden

Usage

- To make checkpoint

```
kill -SIGFREEZE <pid>
```

- To restore checkpoint

```
<binary_name> = recover
```

psncLibCkpt - user interface

1. Changes in source code

```
int
main(int argc, char **argv) {
 printf(„Hello world”);
}
```

```
int
ckpt_target(int argc, char **argv){
 printf(„Hello world”);
}
```

2. Compilation

```
cc –o hello hello.c
```

```
cc –o hello hello.c libckpt.a
   cmain.o –ldl -lrt
```

3. Starting the application

```
./hello
```

```
./hello
```

4. Making a checkpoint

```
ckpt <PID>
```

5. Restarting the application

```
./hello =recover
```

psncLibCkpt - configuration

1. Configuration file .ckptrc (during execution)

```
checkpointing <on|off>
directory <path>
store_files_level <0|1|2|3>
verbose <on|off>
```

2. Configuration file user_config.h (before compilation)

```
#define CKPT_MAXSEM 30
#define CKPT_MAXSEMVECSIZE 10
#define CKPT_MAXMSG 30
#define CKPT_MAXSHM 30
#define CKPT_MAXPROCESSES 256
#define CKPT_MAXPATHSIZE 512
```

psncC/R module

a kernel level checkpointing

Assumptions

- No changes in checkpointed application
  - no code modifications
  - no need for recompiling or re-linking the application
  - no assumptions on programming language, compiler or libraries
- No kernel modifications
  - implementation as module for 64-bit kernel
- Supports both 64- and 32-bit applications
- Minimum impact on operating system

Saved and restored information

- Main process information:
  - memory
    - process memory segments
    - shared libraries segments
  - registers
- Signals information
  - pending signals
  - signal masks
  - signal disposition

Functionality

- Supports open files
  - sequence of used open file descriptors
    - support for regular files, directories, and partially for special files
  - content of opened files
  - position in open files
- Restores current working directory

Functionality (cont.)

- Supports per-LWP (Light Weight Process) information:
  - registers
  - signals
- Process limits

  e.g. maximum number of opened files, maximum amount of memory etc.

Installation

- Install checkpoint module
  - create a checkpoint.conf file in /usr/kernel/drv directory
  - copy compiled module file to /usr/kernel/drv/sparcv9
  - modify file /etc/devlink.tab
  - execute command add_drv checkpoint
- Put chkpnt and resume binaries in globally accessible area and set proper rights

Root privileges required!

Usage

- To make checkpoint

```
chkpnt -p <pid>
```

- To restore checkpoint

```
resume -od <image_directory>
```

Tests

psncC/R tests

- Performance tests
- Three types of applications:
  - our own application (matrix multiplication)
  - application with BLAS mathematic library (matrix multiplication)
  - other applications (basic OS applications)
- Integration with SGE

Performance tests

		Creating process image		
Number of elements	Process image size	Total image creation time	processor usage time(system + user)	I/O operations
1000*1000	16 MB	1,62s	0,77s	52,50%
2000*2000	51 MB	3,58s	1,50s	59,00%
3000*3000	106MB	1m 26s	5,10s	94,00%
9000*9000	950 MB	1m 10s	16,31s	77,00%

"9000*9000" calculations were done on SunFire 6800.

Conclusion:

growing size of application image => the efficiency of storage becomes critical.

Integration with SGE 5.3p3/p4

- **Assumptions**
  - to save process image periodically
  - possible restart from the last saved state
    - by the grid engine
    - outside grid engine

- Submitting a job:

```
qsub -ckpt sge_chkpnt ~/matrix/matrix.sh
```

- Shell script used for running a process:

```
#!/bin/sh
~/matrix/matrix
```

Integration with SGE 5.3p3/p4

Queue `bellis-b-eth.q` of type (qtype):

```
checkpointing batch interactive
```

Checkpoint object:

```
qconf -sckpt sge_chkpnt
ckpt_name          sge_chkpnt
interface          CRAY-CKPT
ckpt_command       ~/bin/chkpnt.sh $job_pid $job_id
migr_command       NONE
restart_command    ~/bin/resume.sh
clean_command      NONE
ckpt_dir           /tmp
queue_list         mistletoe.q
signal             NONE
when               xsm
```

Research issues

- Parallel applications
  - full multithreaded and multiprocess
  - virtual resource identifiers
  - interprocess communication mechanisms
- Process migration
  - virtual resource identifiers
  - interprocess communication mechanisms
- Distributed applications

Functionality in nutshell

Summary

- The checkpoint restart tools for Solaris are now available
- It is easy to distribute and to integrate a dynamically loadable kernel module
- Supporting HPC applications
  - 64 bits applications
- Some work has to be done to support wider range of applications

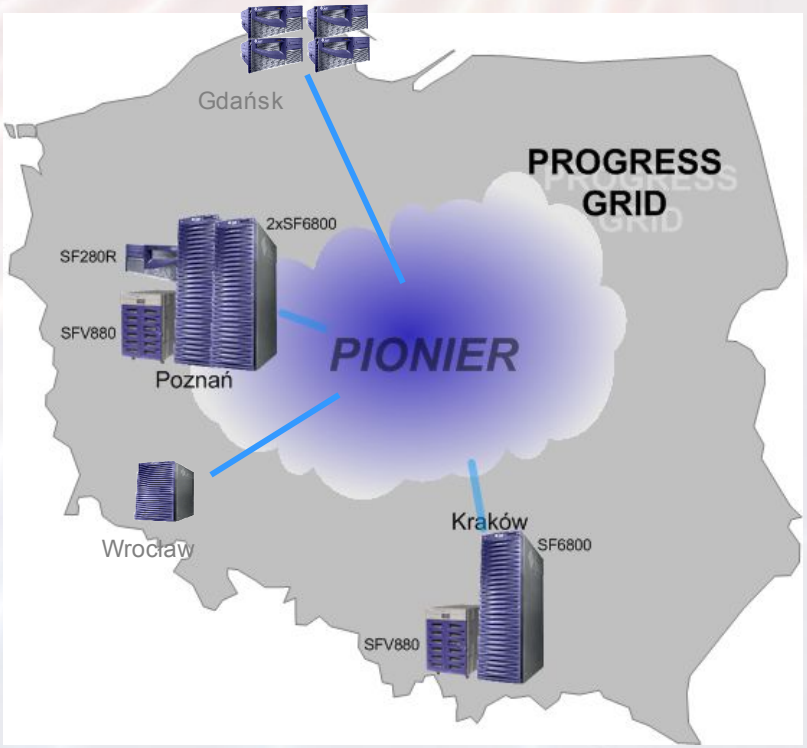
PROGRESS

project no. 6.T11.069.2001C/5688

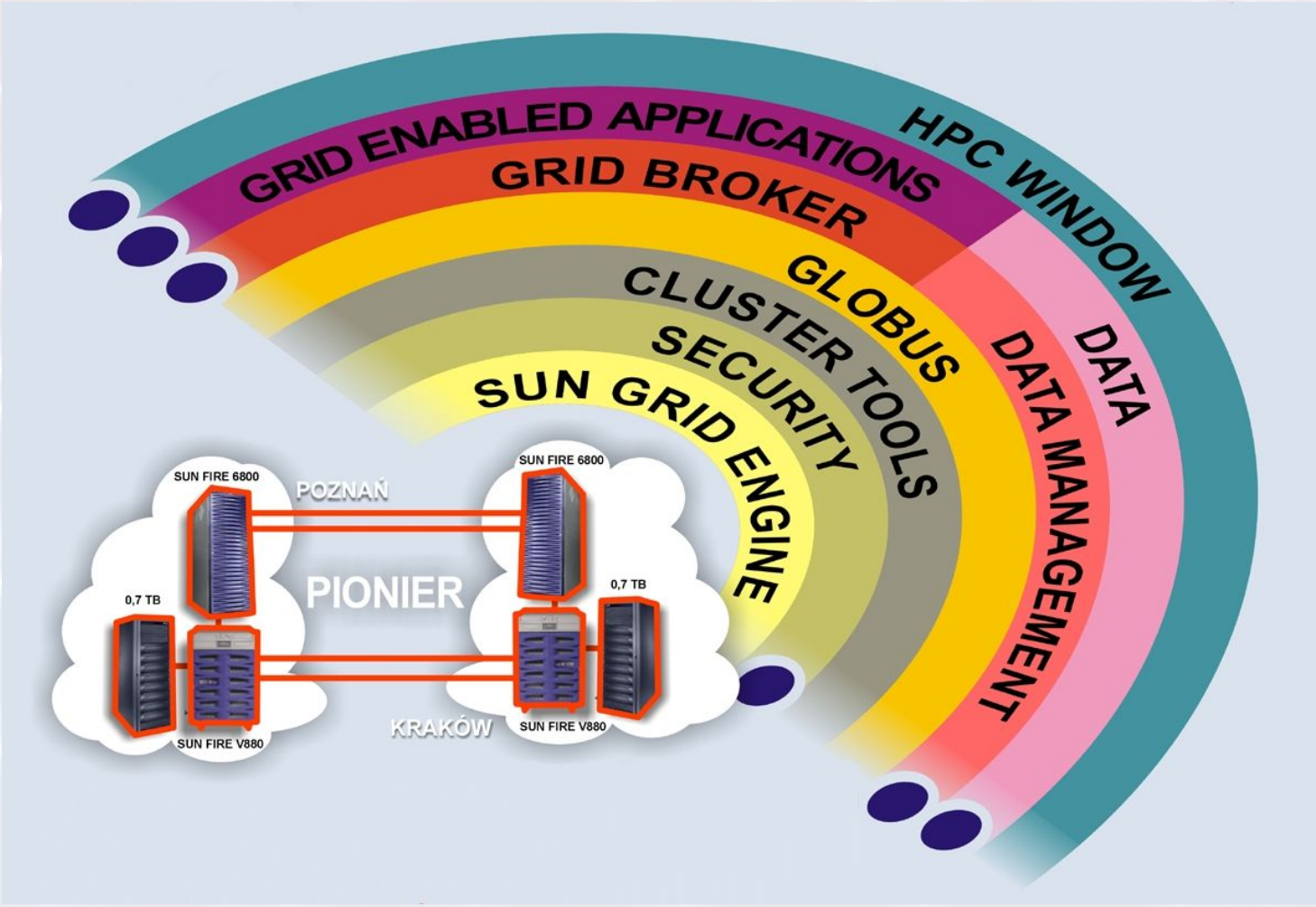
- **Access environment to computational services performed by cluster of SUNs**
- Duration: December 2001 – May 2003
- Deployment: June 2003-December 2003
- Project Partners
  - **SUN Microsystems Poland**
  - **PSNC IBCh Poznań**
  - Cyfronet AMM, Kraków
  - Technical University Łódź
- Co-funded by The State Committee for Scientific Research (KBN) and SUN Microsystems Poland

PROGRESS Grid

- Cluster of 80 processors
- Networked Storage of 1,3 TB
- Software: ORACLE, HPC Cluster Tools, Sun ONE, Sun Grid Engine, Globus
- 2001-2003



PROGRESS Architecture



questions?

meyer@man.poznan.pl

Authors:

Gracjan Jankowski
Radek Januszewski
Rafał Mikołajczak
Maciej Stroiński
Norbert Meyer