Libckpt :
Transparent Checkpointing under Unix

James S. Plank, Micah Beck, Gerry Kingsley,
University of Tennessee ; Kai Li, Princeton University

Proceedings of the USENIX Winter 1995 Technical Conference

Presented by Shian-Tai Chiou
12/13/06

OUTLINE

- Introduction
- Transparent Checkpointing
- User-Directed Checkpointing
- The Mechanics of Checkpointing and Recovery
- Experiments and Results
- Conclusion

Introduction

- Checkpointing is a simple technique for rollback recovery.
- User level
- Transparent Sequential and incremental checkpointing
- User-Directed checkpointing
  little information yield large improvements.

Transparent Checkpointing

- Sequential Checkpointing
- Not completely transparent ：
  change main() to ckpt_target()
- Generates a timer interrupt, and take a sequential checkpoint at each interrupt
- .ckptrc file :
  - checkpointing<on/off>
  - dir <directory>
  - Maxtime <sec>

Transparent Checkpointing

- Incremental Checkpointing
- incremental <on/off>
- old sequential checkpoint file can be deleted. but incremental one can’t.
- Maxfiles <n>
- use page protection to identify which pages should be save.

Transparent Checkpointing

- Forked Checkpointing
- Main-memory checkpointing
- Saving of the checkpoint to disk is overlapped with application execution.
- fork <on/off>
- main-memory or copy-on-write checkpointing

User-Directed Checkpointing

- Memory exclusion
  - Locations is dead
  - Locations is clean
- Heap variables and variables which reside in statically allocated data segment don't work.
- `exclude_bytes(char *addr, int size, int usage)`
  `include_bytes(char *addr, int size)`
- *Usage：* CKPT_READONLY、CKPT_DEAD

User-Directed Checkpointing

- Synchronous Checkpointing
- specify points in the program where it is most advantageous for checkpoint
- checkpoint_here()
- mintime <sec>
- asynchronous – maxtime

  synchronous – mintime

User-Directed Checkpointing

```
main()
{
  struct data *D;
  FILE *fi, *fo;

  D = allocate_data_set();
  fi = fopen("input", "r");
  fo = fopen("output", "w");
  while(read_data(fi, D) != -1) {
    perform_calculation(D);
    output_results(fo, D);
  }
}
```

```
ckpt_target()
{
  struct data *D;
  FILE *fi, *fo;

  D = allocate_data_set();
  fi = fopen("input", "r");
  fo = fopen("output", "w");
  while(read_data(fi, D) != -1) {
    perform_calculation(D);
    output_results(fo, D);
    exclude_bytes(D, sizeof(struct data),
         CKPT_DEAD);
    checkpoint_here();
    include_bytes(D, sizeof(struct data));
  }
}
```

The Mechanics of Checkpointing and Recovery

- Process creation：
  - auto restore text portion of the process's state, and begins execution.
- Recovery routine：
  - process's stack + data segments
- System state restoration：
  - Save the state of open files.
- Processor state restoration：
  - Program counter、stack pointer

The Mechanics of Checkpointing and Recovery

- Save processor state – setjmp()
- Record the state of the open file table
- Data state (program's stack, data segment)

Experiments and Results

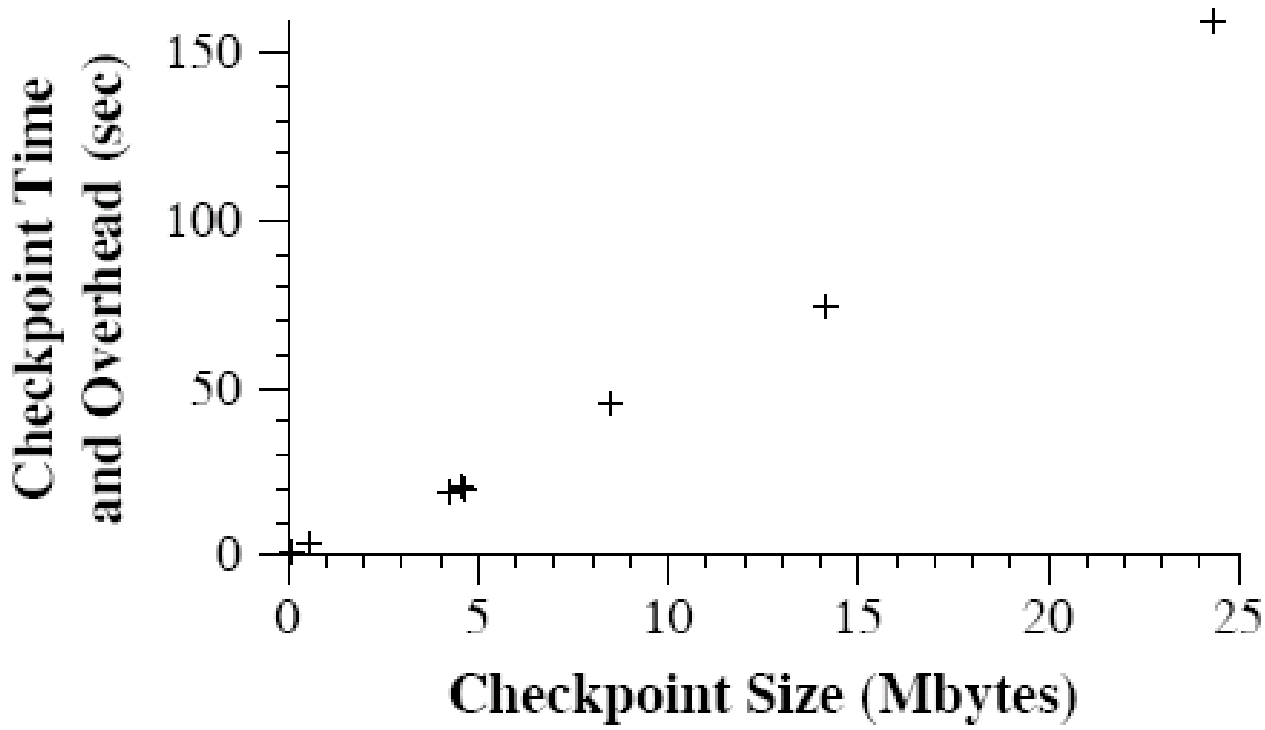
- SunOS

Application	Abbreviation	Language	Running Time (mm:ss)	Maximum Checkpoint Size (Mbytes)	Checkpoint Interval (min)
Matrix Multiplication	MAT	C	15:20	4.6	2
Linear Equation Solver	SOLVE	FORTRAN	13:42	4.6	2
Cellular Automata	CELL	C	17:39	8.4	2
Shallow Water Model	WATER	FORTRAN	25:54	13.1	3
Multicommodity Flow	MCNF	FORTRAN	18:38	24.3	6

Table 1: Description of application instances

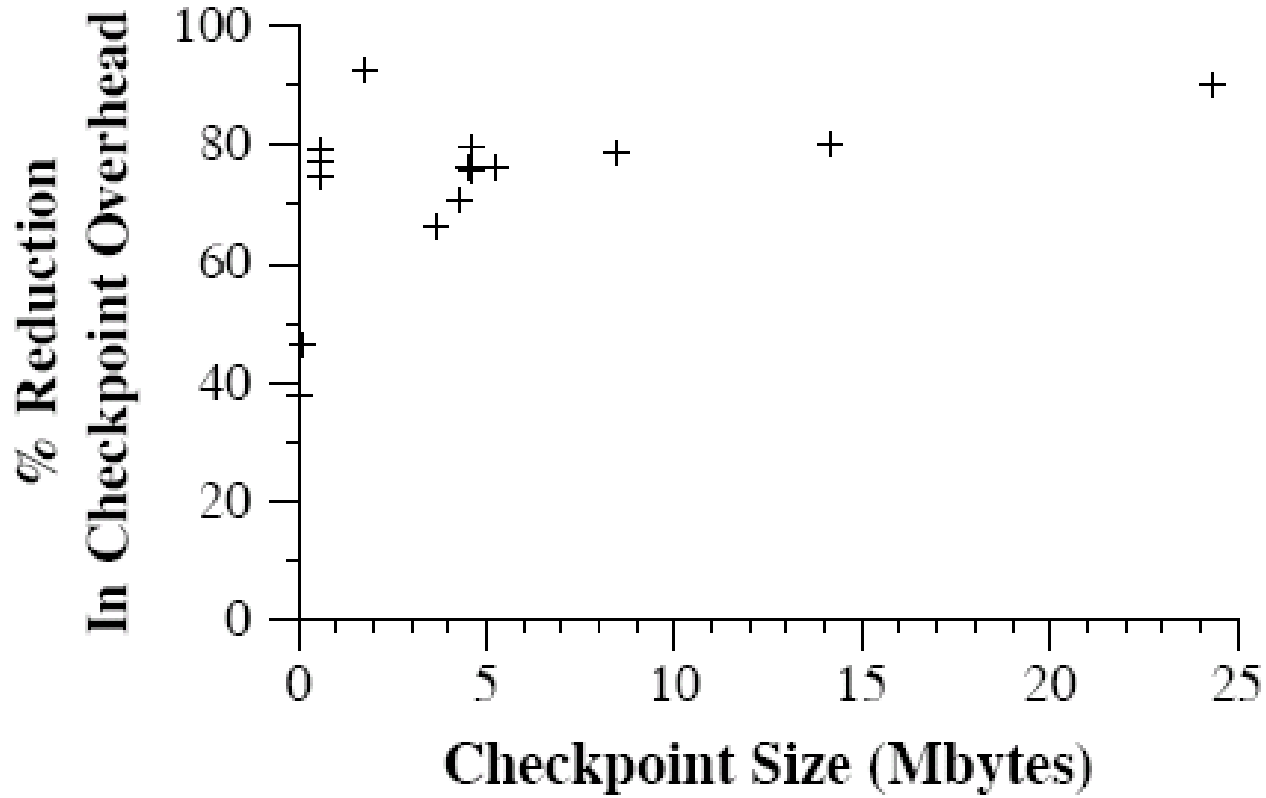
Experiments and Results

- Sequential Checkpointing



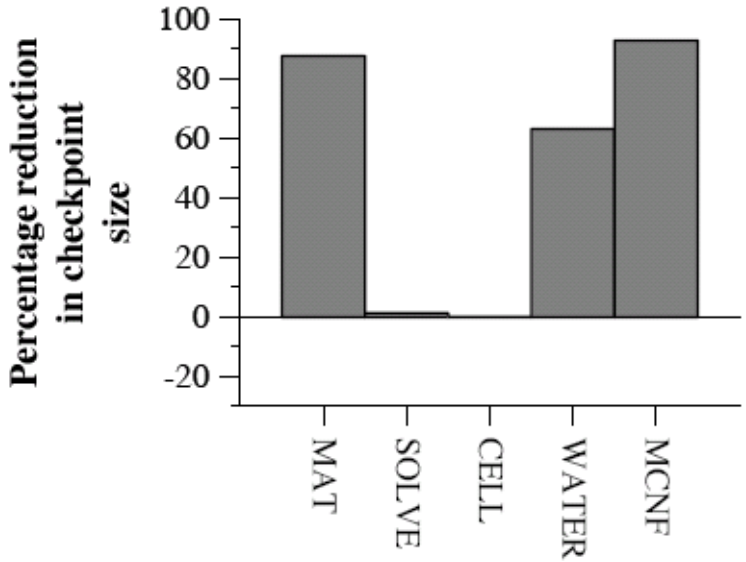
Experiments and Results

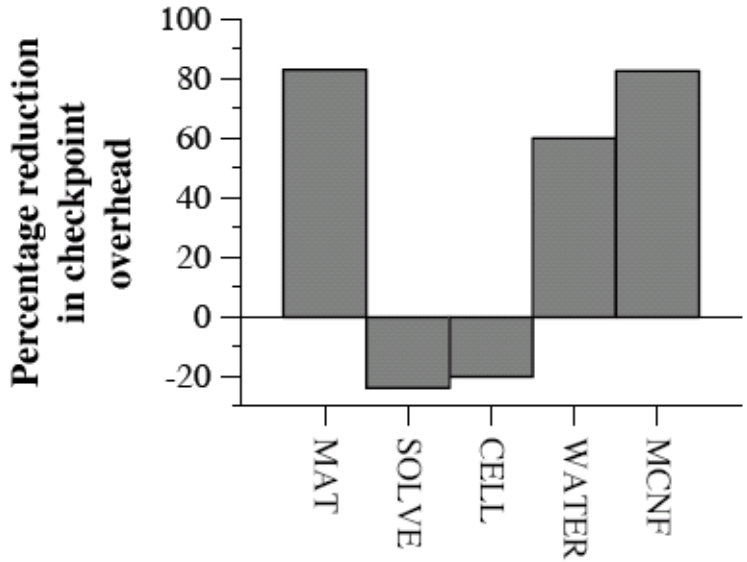
- Checkpointing with fork()



Experiments and Results

- Incremental Checkpointing





Experiments and Results

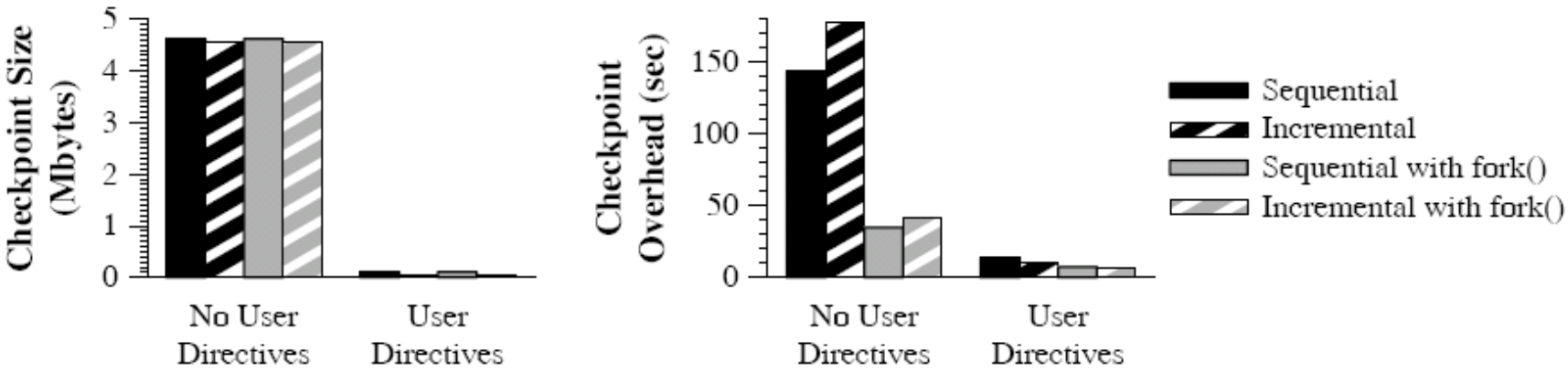


Figure 6: Results of User-Directed Checkpointing on the SOLVE Application

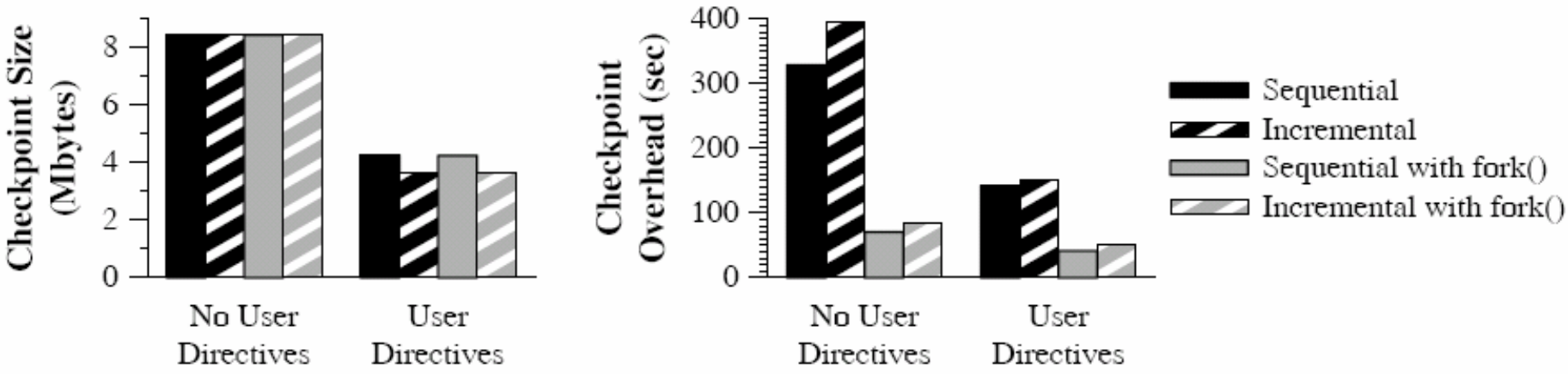


Figure 7: Results of User-Directed Checkpointing on the CELL Application

Experiments and Results

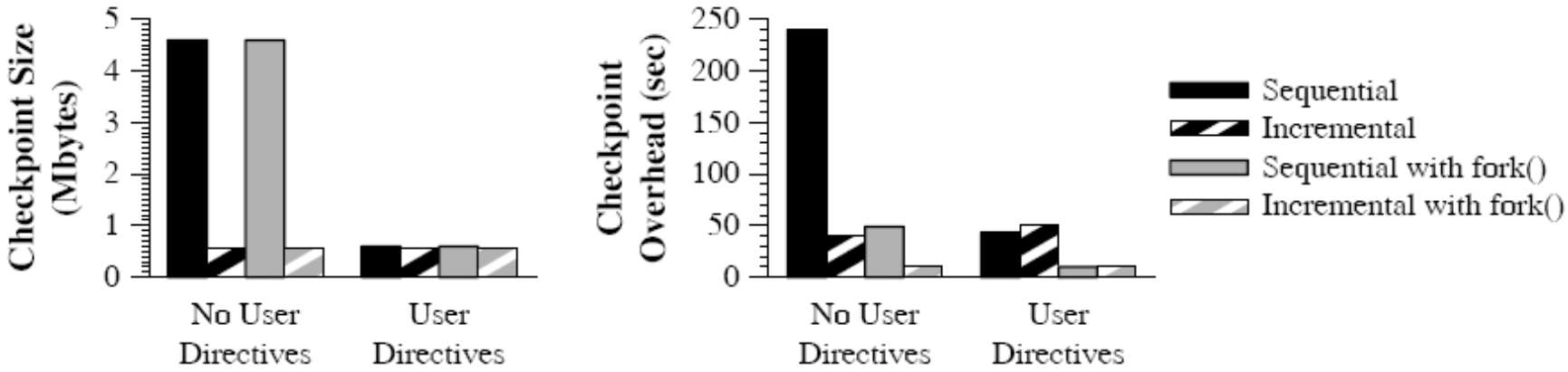


Figure 8: Results of User-Directed Checkpointing on the MAT Application

Conclusion

- Ease of use and low overhead
- Future research ：employ compiler analysis to assist user-directed checkpointing