Overview of Research in the Bernoullli Group

Keshav Pingali, Paul Stodghill
Grigor Bronevetsky, Jim Ezick,
Rohit Fernandes, Daniel Marques,
Kamen Yotov
Department of Computer Science
Cornell University
Outline

• Fault-tolerance
  – Compiler Transformations for Fault-tolerance
  – MPI/FT
  – Other work
• Databases
• Adaptive Compilers
• Collaborations
System-level adaptivity

- Increasingly important for high-performance computing
  - Simulation as the third mode of discovery
  - → explosion of scientific computing
- Adaptive computing
  - Changing resource demands
- Fault-tolerance
  - Better networking
  - → Collaboration, Resource Sharing
  - → Distributed computing
State of the art

- Research in distributed systems
  - General purpose, transparent reliability for user applications
  - Implemented at the (operating) system level
  - Few assumptions about the applications

- Research in restructuring compilers
  - Program analysis and transformations
  - Irregular and parallel applications
Our approach

• Goal:
  – Transparent reliability
  
    *with lower overhead*

• Lower overhead by
  – exploit structures of applications

• Transparent, automatic by
  – using program transformations (ie, compiler aided)

• Compiler + Run-time system = a feasible solution
Current work

• Exploiting determinism
  – Most scientific codes are deterministic (or might as well be)
  – Runtime: Deterministic and Non-deterministic message logging protocols
  – Compiler: insert code to switch between

• Exploiting inherent synchronization
  – Many scientific codes have global synchronization
  – Runtime: Uncoordinated, Block and non-blocking coordinated checkpointing
  – Compiler: find and leverage global synchronization

• Preliminary experiments: workable and practical
Classification of recovery protocols for distributed memory computations

Recovery Protocols
- Check-pointing
  - Uncoordinated: each process saves its state independently of others
  - Coordinated
    - Blocking: hardware/software barrier
    - Non-blocking: distributed snap-shot
- Message-logging: log messages and replay
Program transformations for application-level checkpointing

- Dan Marques
- Program transformations for
  - C + user annotations
  - Globals, Locals, stack
- Future work
  - Heap objects
  - robustness

```c
int fib(int a) {
    int b; int c[5];
    if (a <= 1) {
        return 1;
    } else {
        int temp1;
        int temp2;
        temp1 = fib(a - 1);
        b = temp1;
        /* TAKE-CKPT-HERE */
        temp2 = fib(a - 2);
        b = b + temp2;
    }
    return b;
}
```
int fib(int a)
{
    int b; int c[5];

    if (_CKPT_MODE) {
        // Restore locals from disk
        switch(_CKPT_TMP_LABEL) {
            case 1: goto LABEL_1;
            case 2: goto LABEL_2;
            case 3: goto LABEL_3;
        }
    }

    if (a <= 1)
        return 1;
    else{
        int temp1; int temp2;
        // Push locals to stack'
        LABEL_1:
        temp1 = fib(a - 1);
        // Pop locals from stack'
        b = temp1;
        // Push locals to stack'
        // Checkpoint: write stack' to disk.
        LABEL_2:
        if(_CKPT_MODE){
            _CKPT_MODE = 0;
            fclose(_CKPT_FILE);
        } // Pop locals from stack'
        // Push locals to stack'
        LABEL_3:
        temp2 = fib(a - 2);
        // Pop locals from stack'
        b = b + temp2;
    }
    return b;
Program Analysis for application-level checkpointing

- Jim Ezick and Dan
- If i0..i99 are unchanged at L102, then L0..L99 can be used to reinitialize
- Save minimal data to checkpoint (ie, k)
- Construct recovery script to reinitialize remaining data
- “compiler-theoretic” view of the problem (dominator trees)
- Tradeoff b/w size of checkpoint and recovery cost.

```
L0: i0 = 0;
   .
   .
L99: i99 = 0;
L100: k = 0;
L101: while (k < 1000)
L102:     // check point here
L103:     print f(k,i0,...,i99)
L104:     k = k+1
```
Improved Algorithms for Finding Best Recovery Lines

• Kamen Yotov
• Uncoordinated Checkpointing and (Optimistic) Message Logging
• Simplified the classic algorithm (4 colors to 2)
• Because it’s based on DFS, it’s easier to understand and implement.
MPI/FT

- Sequential checkpointing is working
- MPI “hooks” are implemented
- Engineering difficulty in getting the two to work together.
MPI/FT (cont.)

- Sequential checkpointing architecture
- Challenge – isolate client state from system state

Checkpointing

Master → Client
ReadProcessMemory
"Checkpoint me"

Restore

Master ← Client
WriteProcessMemory
"Resume"
MPI/FT (cont.)

• What should the MPI architecture be?

  Master →

  Client →

  MPI

  Master

  MPI

  Client →

  “Tangled” MPI state

• Poor performance?
Programming with Web Services

- Not implementing the web services themselves (component program)
- Rather, programming with web services as external components (control program)
- Assumptions,
  - A request to a web service can take a very long time to complete.
  - Failures are possible (likely) during the execution of the control program
Dataflow Machine Model

- c.f. Von Neuman
- Machine is collection of computational nodes and dataflow edges
- Values flow along edges as tokens
- Nodes “fire” when tokens available on all in edges
Historical dataflow machines

- **Properties**
  - Inherently parallel
  - Latency tolerance

- **In the literature**
  - Thread models (e.g., fibers)
  - Dependence Flow Graphs (DFG’s)

- **Inspired architectures**
  - Monsoon
  - Tera, Earth/Manna

- **Inspired programming languages**
  - Id, Id Noveau
  - SSISL
Dataflow – the dark side

- Dataflow machines are slow
  - Custom chips
  - Designed for throughput, not peak
- Dataflow languages are hard
  - Scheduling for conventional machines is hard
  - Programmers are used to thinking about state
  - Legacy code without a killer app
Perfect for Web Services

• Control program for web services
  – Latency and fault tolerance is key
  – Efficient scheduling isn’t (b/c latencies are unpredictable)
  – Control programs are small and are novel
Architecture for Programming Web Services

Single Assignment λ-calculus
(Scheme or ML with write-once object)

SAλ Compiler

Dataflow Graph

Service

Service

Service

Dataflow VM

(SOAP or .NET)
Adaptive Compilation

• Empirical Optimization – use experiments to guide parameter selection

• ATLAS – MMM optimized for one cache level
  – Parameters – unrolling, block-size, etc.
  – Experiments run for hours or days

• Memory Hierarchies are very deep (more than 3-levels)
  – Brute-force approach is not practical
Adaptive Compilation (cont.)

• Our approach
  – Use performance models to find neighborhood
  – Use experimentation to find optimal parameter values

• Benefits
  – Much faster than ATLAS-style
  – Therefore can tackle multiple levels of memory hierarchy.

• How to develop models
  – By hand
  – By machine learning
Simulated Cache Misses for MMM

[Graph showing simulated cache misses with different forms and patterns]

Form
- ijk
- ikj
- jik
- jki
- kij
- kij
- (blank)
Discovering cache parameters

- Models are based upon certain cache parameters
  - Cache-size
  - Line-size
  - Associativity
  - Miss penalty

- Remember HW#1?
  - Looking at machine learning and heuristic methods.
History

- 80% of the code of a typical business application deals with **data manipulation** (access, selection, I/O, transformations)
- Only 20% **problem-oriented code**
- Late 60s: **How to reduce the not problem-oriented part?**
Problems w/ Files

• File = dumb sequence of bytes (stream-oriented)
• Change in file format incurs costly source code changes (each function has to “know” the data layout)
• No guarantees for:
  – Data integrity
  – Referential Integrity
• No data manipulation language (DML)
Problems w/ Files (cont.)

- Poor interoperability
- No default support for:
  - Fault tolerance
  - Transactions

Problems solved in current generation RDBMS!

- Microsoft SQL Server
- Informix Dynamic Server (IDS)
- DB2 Universal Database
- Oracle
Solid Models

- Large unstructured data sets (and relations=structure) involved: Topology, Geometry, Mesh(es), Material properties
- Distinct (e.g. pre- and post-processing) phases with different access patterns
- Cannot be effectively handled without a querying language!
Coupled Thermo-Mechanical Simulation

• Heat conduction
  – Heat fluxes on the central hole from MSU
  – Fixed temperature (500K) for the cooling holes

• Solve for Temperature, spawn off wavelet analysis

• Coupling
  – Thermal stresses
  – Temperature dependent Young’s modulus and Poisson ratio

• Deformation
  – Pressures and shear stresses on the inner hole from MSU
  – Fixed displacement on one end surface

• Solve for displacement
SELECT A.m_tet_id
FROM
(SELECT m_tet_id
    FROM MVerticesOfMTetrahedron
    WHERE m_vertex_id IN
        (SELECT m_vertex_id
            FROM MVerticesOfMTrianglesOnTSurface
            WHERE m_triangle_id IN
                (SELECT m_triangle_id
                    FROM MSU_wall_conditions)
        )
    ) AS A
JOIN
    TPartitioning AS B ON A.m_tet_id = B.m_tet_id
WHERE B.partition = 'my_MPI_rank';
Adaptivity

- **Modeling**
  - Coupling

- **Algorithmic**
  - Identify hotspots and stress concentrations
  - Explore different discretization techniques

- **“FEM backend”**
  - H- and P-adaptivity
  - Different solvers and preconditioners

- **Database**
  - Query granularity
Advantages

• Database preserves a global view in a distributed simulation
• Reduced code size (SQL statements = strings, ODBC calls)
• The power of SELECT
• Interoperability (ODBC, OLE DB, ADO, HTTP, XML, …)
• Higher concurrency
Collaborations

• Cornell
  – Paul Chew, Steve Vavasis, Bart Selman, Carla Gomes
  – Cornell Fracture Group, Civil Engineering
  – Cornell Theory Center
• Engineering Research Center, Mississippi State University
• College of William and Mary
• Dept of Comp Sci, UIUC
• IBM TJW