ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

CSCS
Swiss National Supercomputing Centre

Resiliency in massively parallel quantum simulations

R. Solca[1], G. Bosilca[2], R. Graham[3], P. Staar[1], A. Kozhevnikov[1], and Thomas C. Schulthess

(1) ETH Zurich; (2) Univ. of Tennessee; (3) Oak Ridge National Laboratory

Outline

- Introduction
- Monte Carlo integration
- Domain decomposition integration
- first application: DCA++
  - overview
  - results
- second application: DRC

Overview of the problem

- Node failures on massively parallel systems
  - As the number of nodes increases, individual components have to be more reliable (and expensive), or we can try to make the software system, and thus the entire system, resilient to component failure

- This talk is about creating simulations systems that are resilient to component failure.

What is it that we can expect?

- We have (at least) three possibilities
- The system catches and corrects errors at a very low level
  - none of the library or application level software has to be aware of resilience
- Components of the system (e.g. nodes) can fail and the software stack has to catch and deal with the problem at the OS or library level
  - even here the are some fundamental problems – does the application have to know that it is using a fault resilient version of a library?
- But not all codes use libraries and libraries may not be able to deal with failures, hence application have to deal with problems and recover to be resilient

Two versions of MPI that support fault resilience

- Fault Tolerant MPI (FT-MPI)
  - Graham E. Fagg, Edgar Gabriel, George Bosilca, Thara Angskun, Zizhong Chen, and Jelena Pjesivac-Grbovic, *Extending the MPI Specification for Process Fault Tolerance on High Performance Computing Systems*, In Proceedings of the ISC2004 conference, Heidelberg, 2004.
- Run-through Stabilization
  - Joshua Hursey, *Run-through Stabilization User Guide*, https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/run_through_users_guide, 2011
  - Joshua Hursey, Richard Graham, Greg Bronevetsky, Dario Buntinas, and Adam Moody, *Run-through Stabilization Interfaces and Semantics*, https://svn.mpi-forum.org/trac/mpi-forum-web/wiki/ft/run_through_stabilization, 2011

Algorithm-based fault tolerance of matrix operations

K. Huang and J. A. Abraham, IEEE Trans. Comput. **33**, 518-528 (1984)
G. Bosilca, R. Delmas, J. Dongarra, and J. Langou, J. Par. Distrib. Comput. **69**, 410-416 (2009)

Outline

- Introduction
- Monte Carlo integration
- Domain decomposition integration
- first application: DCA++
  - overview
  - results
- second application: DRC

Let's look at a simple problem: Monte Carlo

Algorithm 1 General Montecarlo algorithm

```
root prepares data
Broadcast(data)
Compute local results (each process has an independent Markov chain)
Allreduce(results)
average = results / (processes · measurement per process)
output the results
```

We use collective to find out about the state of the system

Default broadcast (uses tree algorithm)

Default broadcast – doesn't work for FT Monte Carlo

Default broadcast – doesn't work for FT Monte Carlo

Default allreduce (uses butterfly algorithm)

Default allreduce – doesn't work either ...

Broadcast that is usable in FT Monte Carlo method

Use functions supplied by FT communication library to check state of system

> MPI_Barrier in FT-MPI

> MPI_Validate_All in run-through stabilization

Algorithm 2 Fault tolerant broadcast

```
1:  choose a root
2:  if this process is the root then
3:      prepare data
4:  end if
5:  repeat
6:      Broadcast(data)
7:      Check
8:      if Failure then
9:          Recover
10:         if root failed then
11:             GoTo 1
12:         end if
13:     end if
14: until Success
```

Allreduce that is usable in FT Monte Carlo method

Algorithm 3 Fault tolerant average of results

```
repeat
    Allreduce(results)
    Check
    if Failure then
        Recover
    end if
until Success
average = result / (alive processes · measurements per process)
```

FT Monte Carlo method

Algorithm 1 General Montecarlo algorithm

1: root prepares data
2: Broadcast(data)
3: Compute local results (each process has an independent Markov chain)
4: Allreduce(results)
5: average = results / (processes · measurement per process)
6: output the results

Cannot use default broadcast and allreduce, use customized versions instead

FT Monte Carlo method: Theoretical analysis

- Montecarlo error with p processes

$$\mathrm{err}_p = \frac{\sigma}{\sqrt{n_{tot}}} \propto \frac{1}{\sqrt{n_p p}}$$

- In case of some process failures
  - With FT algorithm the error becomes

$$\mathrm{err}_{p(1-f)} = \sqrt{\frac{p}{p(1-f)}}\mathrm{err}_p \approx \left(1 + \frac{f}{2}\right)\mathrm{err}_p$$

  - With standard algorithm: NO results

Error behavior in case of failures

Outline

- Introduction
- Monte Carlo integration
- Domain decomposition integration
- first application: DCA++
  - overview
  - results
- second application: DRC

Second problem: domain decomposition integration (no data recovery needed)

In the case we do not want to rebuild died processes

Algorithm 1 Domain decomposition

```
root prepares data
Broadcast(data)
divide the domain among the processes
compute the local part of the integral
Allreduce(result)
output the results
```

We use collective to find out about the state of the system

FT domain decomposition

First possible way:

Algorithm 5 Fault tolerant domain decomposition: first way

```
while integration not completed do
    divide the remaining interval among the alive processes
    compute the local part of the integral
    repeat
        Allreduce(partial_result)
        Check
        if Failure then
            Recover
        end if
    until Success
    result += partial_result
end while
```

The intervals of the failed processes is divided between the remaining processes

Drawback: need reduction at each cycle

FT domain decomposition

Second possible way:

Algorithm 6 Fault tolerant domain decomposition: second way

```
divide the interval among the alive processes
compute the local part of the integral
repeat
    Allreduce(result)
    Check
    if Failure then
        Recover
    end if
until Success
for all dead processes do
    compute the integral over local interval
    result += local_result
end for
```

The intervals of the failed processes is computed by each process

Drawback: more computation

Outline

- Introduction
- Monte Carlo integration
- Domain decomposition integration
- first application: DCA++
  - overview
  - results
- second application: DRC

Superconductivity: a state of matter with zero electrical resistivity

Heike Kamerlingh Onnes (1853-1926) Discovery 1911

Microscopic Theory 1957

PHYSICAL REVIEW VOLUME 108, NUMBER 5 DECEMBER 1, 1957

Theory of Superconductivity*

J. Bardeen, L. N. Cooper,† and J. R. Schrieffer‡
Department of Physics, University of Illinois, Urbana, Illinois
(Received July 8, 1957)

A theory of superconductivity is presented, based on the fact that the interaction between electrons resulting from virtual exchange of phonons is attractive when the energy difference between the electrons states involved is less than the phonon energy, $\hbar\omega$. It is favorable to form a superconducting phase when this attractive interaction dominates the repulsive screened Coulomb interaction. The normal phase is described by the Bloch individual-particle model. The ground state of a superconductor, formed from a linear combination of normal state configurations in which electrons are virtually excited in pairs of opposite spin and momentum, is lower in energy than the normal state by amount proportional to an average $(\hbar\omega)^2$, consistent with the isotope effect. A mutually orthogonal set of excited states in one-to-one correspondence with those of the normal phase is obtained by specifying occupation of certain Bloch states and by using the rest to form a linear combination of virtual pair configurations. The theory yields a second-order phase transition and a Meissner effect in the form suggested by Pippard. Calculated values of specific heats and penetration depths and their temperature variation are in good agreement with experiment. There is an energy gap for individual-particle excitations which decreases from about $3.5kT_c$ at $T=0°K$ to zero at T_c. Tables of matrix elements of single-particle operators between the excited-state superconducting wave functions, useful for perturbation expansions and calculations of transition probabilities, are given.

Superconductivity in the cuprates 1986

Quantum cluster theories

Maier *et al.*, Rev. Mod. Phys. '05

On-site Coulomb repulsion (~A)

Antiferromagnetic correlations / nano-scale gap fluctuations

Gomes et al. (2007)

Superconductivity (macroscopic)

Explicitly treat correlations within a localized cluster

Treat macroscopic scales within mean-field

Coherently embed cluster into effective medium

First application: DCA++

Algorithm 1 DCA++

```
root prepares data
Broadcast(data)
for it = 1 → N_iterations do
    compute some integrals with domain decomposition
    Monte Carlo integration
    average results
end for
output the results
```

Use previous strategies to include fault tolerance

FT-DCA++ Results

- No performance results:
  - Test on small clusters (20-100 processes per run).
- Self energy after 12 iterations

60 processes

60 processes
30 killed in 30th iteration

Outline

- Introduction
- Monte Carlo integration
- Domain decomposition integration
- first application: DCA++
  - overview
  - results
- second application: DRC

Overview of the application

Bare Coulomb interaction

$$w(\vec{r},\vec{r}') = \frac{e^2}{|\vec{r}-\vec{r}'|}$$

Screened Coulomb interaction

$$W(\vec{r},\vec{r}',\omega)$$

The challenge: $W(\vec{r},\vec{r}',\omega)$ is extremely expensive to compute

Can this be computed at scale and efficiently?

The computationally intensive part – lots of nested loops

$$\chi^{KS}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \frac{1}{N_k\Omega}\sum_{\mathbf{k}}^{BZ}\sum_{jj'}\langle\psi_{j\mathbf{k}}|e^{i(\mathbf{G}+\mathbf{q})\mathbf{r}}|\psi_{j'\mathbf{k}+\mathbf{q}}\rangle\times$$

$$\times\frac{f_{j\mathbf{k}}-f_{j'\mathbf{k}+\mathbf{q}}}{\epsilon_{j\mathbf{k}}-\epsilon_{j'\mathbf{k}+\mathbf{q}}+\omega+i0^+}\langle\psi_{j'\mathbf{k}+\mathbf{q}}|e^{-i(\mathbf{G}'+\mathbf{q})\mathbf{r}}|\psi_{j\mathbf{k}}\rangle$$

$$\chi^{KS}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \frac{1}{N_k\Omega}\sum_{\mathbf{k}}^{\mathbf{BZ}}\sum_{\beta}A^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}}B^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}'}(\omega)$$

Reduce to a complex matrix multiply – BLAS3 zgemm
(code rewrite yields order of magnitude improvement in time to solution)

Parallelize with MPI-Grid

$$\chi^{KS}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \frac{1}{N_k \Omega} \sum_{\mathbf{k}}^{\mathbf{BZ}} \sum_{\beta} A^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}} B^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}'}(\omega)$$

Execution time of DRC code computing W (and U) for La_2CuO_4

Parallelize with MPI-Grid

$$\chi^{KS}_{\mathbf{G}\mathbf{G}'}(\mathbf{q},\omega) = \frac{1}{N_k \Omega} \sum_{\mathbf{k}}^{\mathbf{BZ}} \sum_{\beta} A^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}} B^{\mathbf{k},\mathbf{q}}_{\beta\mathbf{G}'}(\omega)$$

Algorithm 1 General workflow

```
build communicators
initialize wave-function
partition q-vectors over z-direction of the MPI grid
partition frequency points over x-direction of the MPI grid
for q = 1 → N_q^loc do {loop over local fraction of q-vectors}
    partition k-vectors over x-direction of the MPI grid
    partition interband transitions over y-direction of the MPI grid
    compute local fraction of matrix elements
    for ω = 1 → N_ω do {loop over all frequency points}
        compute local fraction of χ^KS_GG'(ω, q)
        for each xy-plane reduce the full χ^KS_GG'(ω, q) on the process who handles frequency ω
    end for
    for ω = 1 → N_ω^loc do {loop over local fraction of frequency points}
        compute local fraction of W_GG'(ω, q)
        compute contribution to U^T_nn'(ω) matrix
    end for
end for
reduce full U^T_nn'(ω) matrix
output U^T_nn'(ω) matrix to files
```

Recover data of failed processes

- Need redundant storage
- Strategy:
  - Extra processes that stores negative checksums
  - Example: one extra process
    - Checksum: $$x_{p+1} = -\sum_{i=1}^{p} x_i$$
    - Recovery: $$x_f = -\sum_{\substack{i=1 \\ i \neq f}}^{p+1} x_i$$
    - Can recover only one failure

Change in the MPI Grid

- Add a process in the x-dimension
- Two different grids:
  - computational collectives
  - checksum/recovery collectives
- No extra processes in
  - z-dimension: xy-planes almost independent (reduction at the end)
  - y-dimension: has only 1-10 processes
- Drawback: 2 failed processes in the same x-line

Fault resilient DRC code

Algorithm 3 Fault tolerant computation of matrix elements and $\chi^{KS}_{GG'}(\omega, q)$

```
if matrix elements have to be recovered then
    recover local fraction of matrix elements
else
    repeat
        Compute local fraction of matrix elements
        checksum matrix elements
        if failure then
            recover application
        end if
    until success
end if
if χ^KS_GG'(ω,q) have to be recovered then
    recover χ^KS_GG'(ω,q)
end if
for ω = i_2 → N_ω do {loop over all frequency points}
    compute local fraction of χ^KS_GG'(ω,q)
    repeat
        for each xy-plane reduce the full χ^KS_GG'(ω,q) on the process who handles frequency ω
        send χ^KS_GG'(ω,q) to the corresponding ft-process
        if failure then
            recover application
        else
            ft-processes updates the checksum of χ^KS_GG'(ω,q)
        end if
    until success
end for
```

Algorithm 2 Fault tolerant workflow

```
build communicators
if process is respawned then
    get application state
end if
initialize wave-function
partition q-vectors over z-direction of the MPI grid
partition frequency points over x-direction of the MPI grid
if local U^T_nn'(ω) have to be recovered then
    recover local U^T_nn'(ω)
end if
for q = i_1 → N^loc_q do {loop over local fraction of q-vectors}
    partition k-vectors over x-direction of the MPI grid
    partition interband transitions over y-direction of the MPI grid
    compute matrix elements and χ^KS_GG'(ω,q) {Algorithm 3}
    for ω = i_3 → N^loc_ω do {loop over local fraction of frequency points}
        compute local fraction of W_GG'(ω,q)
        compute contribution to U^T_nn'(ω) matrix
        repeat
            checksum contribution U^T_nn'(ω)
            if failure then
                recover application
            else
                update local U^T_nn'(ω)
            end if
        until success
    end for
end for
repeat
    reduce full U^T_nn'(ω) matrix
    if failure then
        recover application
    end if
until success
repeat
    output U^T_nn'(ω) matrix to files
    if failure then
        recover application
    end if
until success
```

1. call MPI recovery function
2. rebuild all communicators
3. recover data

Summary and conclusion

- Fault resilient (production) versions of DCA++ and DRC
- Monte Carlo based codes (e.g. DCA++):
  - FT-MPI works well, but may not be optimal in terms of performance
  - run-through stabilization is more efficient as the communicators are only fixed, and there is no need to rebuild the communicator
- Distributed matrix based codes (e.g. DRC)
  - run-through stabilization currently doesn't work, as we need to rebuild processes (generally true for domain decomposed problems)
  - FT-MPI works well, but need to rebuild all the communicators
- We do not yet have performance measurements at scale
  - these will be provided soon as we install fault tolerant version of MPI on larger CSCS machines
- We want our application to produce results, not error codes!
  - All codes and algorithms have to be build with FT in mind from the ground up!
  - We need a standard for FT in MPI sooner rather than later

QUESTIONS / COMMENTS?