Zen のクラスタ並列化

加 藤 英 樹[†1] 竹 内 郁 雄[†1]

Zen に粗粒度のネットワーク並列を実装した．ソースコードの追加・修正は千行足らずで済んだ．並列化には，現在のネットワーク並列囲碁ソフトのほとんどが採用している，粗粒度のルート並列方式を用いたが，MPI は用いなかった．これにより，HPC クラスタだけでなく，MPI を実装していない，あるいはできない，PC やゲームコンソールなどから成るクラスタでも動く．また，LAN だけでなく，WAN でも動かすことができるし，運用中ノードコンピュータを動的に切り離したり繋いだりすることもできる．諸般の事情から，性能はまだ測定できてない．

Running "Zen" on Computer Clusters

Hideki Kato [†1] and Ikuo Takeuchi[†1]

We have implemented a network parallelism on "Zen." Less than one thousand lines are added or modified. Coarse grain root parallel algorithm which current almost all network parallel MCTS programs are using is used. MPI is, however, not used in our implementation. This allows our program can run on the computer clusters consist of a mixture of board servers, personal computers and game consoles. By various reasons, the performance is not measured.

1. はじめに

モンテカルロ探索（Monte-Carlo tree search; 以下 MCTS）を用いた囲碁ソフトは，HPC クラスタ上の MoGo (以下 MoGoTitan[⋆1]) が公式戦で七子置いて九段のプロ棋士から勝ち星を挙げるまでになった．[⋆2]

我々はこれまでクライアントサーバ型細粒度並列方式を研究してきた[1),2)]が，今回，今年 5 月の Computer Olympiad Pamplona[⋆3] の 19 路碁で優勝した Zen に粗粒度のネットワーク並列を実装し，大規模な HPC クラスタと小規模な PC クラスタの 2 種類のプラットフォームに移植したので報告する．

本論文の構成は，本節が導入，2 節で既存研究を紹介し，3 節は本研究の目的と課題，4 節で具体的な実装について述べ，5 節は実験と評価，6 節はまとめである．

2. 既存研究

MCTS のクラスタ向き並列化に関する報告は T. Cazenave ら[3)] によるものが最初で，single-run, multiple-runs, at-the-leaves の 3 通りの並列 MCTS を，Intel Pentium4 PC 16 台の MPI クラスタに実装し，比較・評価した．彼らはその後，[4)] 非対称構成の MPI クラスタでの実験結果も報告している．S. Gelly ら[5)] は SMP システムおよび MPI クラスタでの並列化を議論し，MoGoTitan が用いているアルゴリズムについて，簡単な性能評価を行った．古典囲碁ソフトの強豪 Many Faces of Go の作者 D. Fotland は，Computer Olympiad Beijing[⋆4] に，HPC クラスタで走る，MCTS を導入した Many Faces of Go で参加し，9 路と 19 路の両方で優勝した．M. Muëller6) は，Computer Olympiad Pamplona の 9 路で優勝した Fuego の実戦記の中で，クラスタ版 Fuego に関して報告している．Many Faces of Go と Fuego は，基本的に MoGoTitan と同じアルゴリズムを用いていると両作者は言っており，これが事実上の標準となっている．なおこのアルゴリズムは，T. Cazenave

†1 東京大学大学院情報理工学系研究科創造情報学専攻
The Department of Creative Informatics, The Graduate School of Information Science and Technology, The University of Tokyo

⋆1 http://www.cs.unimaas.nl/g.chaslot/muyungwan-mogo/

⋆2 Human-Computer Go Challenges (http://www.computer-go.info/h-c/index.html).

⋆3 http://www.grappa.univ-lille3.fr/icga/event.php?id=41&lang=3

⋆4 http://www.grappa.univ-lille3.fr/icga/event.php?id=37&lang=3

ら[3] が multiple-runs, あるいは G. Chaslot ら[7] が root-parallelization と呼んでいるものに属す.

我々も今回このアルゴリズムを採用したが, MPI は使わなかった. これは, peer-to-peer, broadcast, multicast 等の複数の通信方式による性能の比較を可能にしたいということも理由の一つであるが, 専用の HPC クラスタだけでなく, PC や, MPI を実装できないゲームコンソール等をそれほど高速でない LAN, あるいはインターネットの様な WAN で結合した環境, さらに先の展開として, ロボットや Intelligent Vehicle の様に, 多数の組み込み用プロセッサを比較的低速な LAN で結合した環境の様な, 非常に広い範囲のコンピュータクラスタで動作させたいということが一番の理由である.

3. 目的と課題

MCTS アルゴリズムをネットワーク並列化する時の課題として以下の 3 つが挙げられる.

(1) スケーラビリティ: 台数の増加と共に性能がリニアに増加することが望ましい.

(2) コスト: ノード数が多いのでできるだけ安価に構成できることが望ましく, システム構成の自由度が高いことはこれを助ける. 特に, HPC クラスタのコストのかなりの部分を占める高価な超高速ネットワーク I/F ではなく, 廉価で一般的な Gigabit Ethernet (GbE) を使ったコンピュータクラスタでどの程度性能が低下するかは興味深い.

(3) 耐故障性: ノード数の増加に伴い故障する確率が増加するので, 故障耐性が高いことが望ましい. また, 動作中に各コンピュータを切り離したり, 途中から参加させられることが, 特にインターネットなどの信頼性の低い WAN の場合, 強く望まれる.

(4) さらに, 本報告特有の項目として, ネットワーク並列化のために追加・修正するソースコードの量がある. これは, 今回作成したプログラムのネットワーク周りのコードを公開し, 他の MCTS 囲碁ソフトを簡単にネットワーク並列化できるようにすることも意図しているからである.

本論文の目的は, 超高速ネットワークを用いた大規模な HPC クラスタと, GbE を用いた小規模な PC クラスタ上での, クラスタ並列版 Zen の性能の測定と評価, および並列化に必要な作業量の実測である.

4. 実　　装

4.1 プラットフォーム

対象プラットフォームは, 大規模な HPC クラスタ

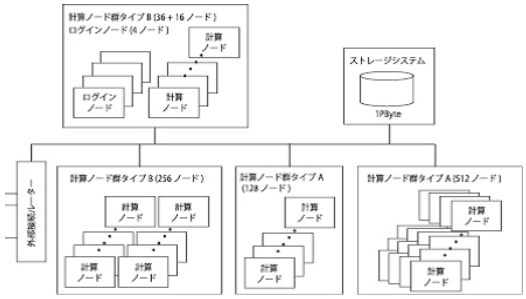

図 1 HA8000 のシステム構成 (HA8000 のウェブサイトより引用).

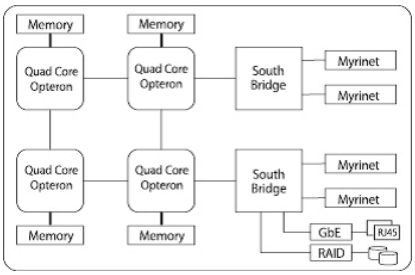

図 2 HA8000 のノード構成 (HA8000 のウェブサイトより引用).

表 1 HA8000 の諸元.

ノード数	952
インタコネクト	Myri 10G
CPU (1 node)	4 × AMD quad core Opteron (2.3 GHz)
Memory (1 node)	32 GB

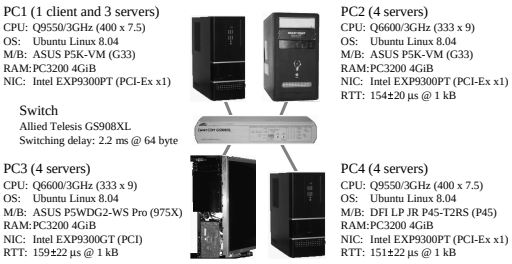

図 3 自作 PC クラスタ.

として東京大学情報基盤センターの T2K スーパーコンピュータ, HA8000 システム (**図 1**) [⋆1], 小規模な PC クラスタとして 4 コアの自作 PC 4 台を GbE LAN で結合したクラスタ (**図 3**) の 2 種類である. HA8000 システムの OS は RedHat Linux, 自作 PC クラスタの OS は Ubuntu Linux である.

Zen は Windows 環境で C++を用いて開発されて

⋆1 http://www.cc.u-tokyo.ac.jp/service/ha8000/, http://journal.mycom.co.jp/articles/2008/06/15/t2k/index.html の記事が詳しい.

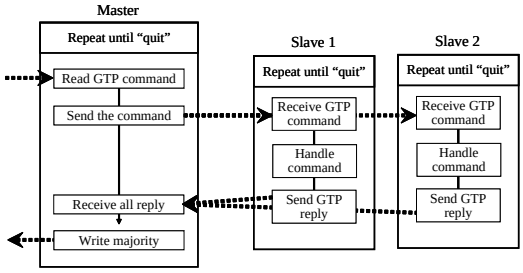

図 4 GTP コマンドの処理

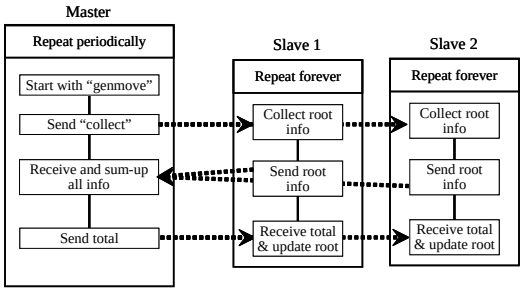

図 5 探索木中の情報の交換

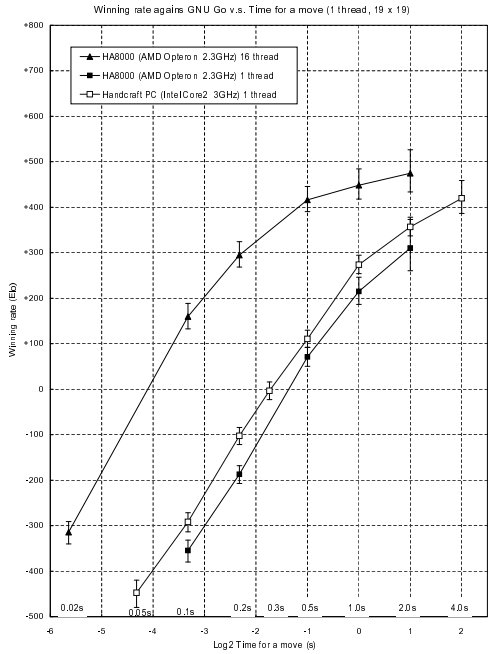

図 6 GNU Go に対する勝率 (19 路).

いるが, Linux も意識して作られているため, Linux への移植には大きな手間はかからなかった. 今回の並列化の要であるネットワーク周りには, ソースコードの Windows との共用を意識して, C++の Boost ライブラリの asio パッケージを利用した. また, ソースコードの修正・追加ができるだけ少なくなること, 他の MCTS を用いたプログラムのクラスタ並列化に利用できることを意識して作成した. 追加したコードは千行弱, 修正した行数は 100 行以下である.

4.2 アルゴリズム

2 節のアルゴリズムは対称マルチスレッドをベースにしてゼロ番のスレッドだけ特別扱いにしているが, 我々はマスタースレーブ構成を採用した. これは, モジュールの独立性を高くして Zen のソースコードの修正をできるだけ少なくしたかったためである.

マスターとスレーブのの動作の概要を**図 4** と**図 5** に示す.

マスターは外部との入出力を司り, 標準入力から読んだ GTP コマンドを全スレーブに分配し, 返された返事の中から単純多数決で一つを選んで標準出力に書き出す. これにより, 将棋ソフトの文殊 [⋆1] が用いている合議アルゴリズムと類似した効果が期待できる可能性がある. また, 探索中, マスターは各スレーブの探索木の中の情報を定期的に集計し送り返す. これは探索スレッドとは別のスレッドとして動作する. このモジュールは探索木の中の情報にアクセスするため, マスターが別プログラムであることと対照的に, Zen の中に組み込む必要がある.

表 2 PC クラスタでの勝率. 1 秒/手, 各 1 コア使用.

路数	PC 台数	対戦相手	WR
9	3	自己	45.5±2.2% (-32±15 Elo)
9	4	自己	52.0±6.9% (+14±48 Elo)

マスター・スレーブ間の通信には TCP/IP による peer-to-peer 方式を用いた. Peer-to-peer による 1 対全通信は UDP より遅延時間が長くなるが, UDP の使用はセンターの様な共用環境ではあまり望ましくなく, インターネットの様な WAN では途中のルータを超えられない可能性がある. また multicast は IPv4 では使用に不安があるので TCP を採用したが, いずれ, IPv6 への変更と共に multicast を採用する予定である. しかし, 一回の通信量がそれほど多くない (19 路で $48 \times 19^2 \times N \times 2$ バイト; N はノード数) ので, 実際にはそれほど性能に影響はないと見込んでいる.

5. 実験と評価

プラットフォームとプロセッサ数 (=スレッド数) を変えて, 対 GNU Go, および自己対戦の勝率を測定

⋆1 http://homepage1.nifty.com/ta_ito/ito-lab/gougi/gougi.html

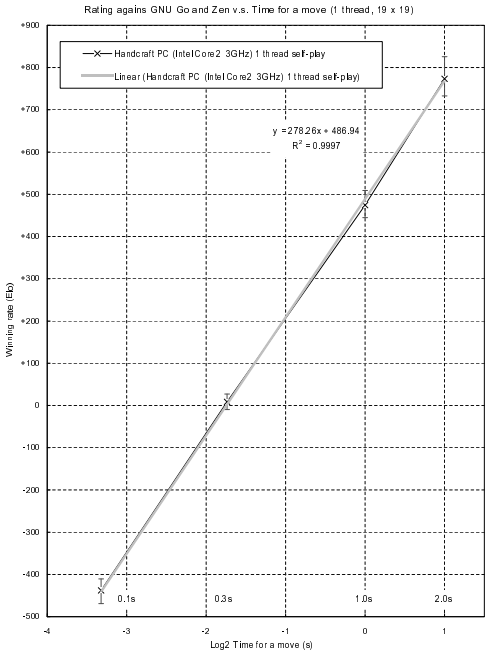

図 7 Zen に対する勝率 (19 路).

した.

まず, 同じ強さになる思考時間の比 (7) の *s*trength speedup) で台数効果を測定するために, 1 スレッドと 16 スレッドで一手当たりの時間を順次変更した強さを測定した. 強さの基準となる対戦相手には, 広く使われている GNU Go を用いた (**図 6**, **図 8**). 19 路の 16 スレッドの勝率を見ると, 400 Elo 辺りから大きく曲がって (飽和して) おり, これより上の範囲は測定できない. これでは *s*trength speedup の測定に使う基準として, ダイナミックレンジも狭くあまり宜しくない.

続いて Zen と Zen との自己対戦の勝率を測定した (**図 7**, **図 9**). こちらは, 特に 19 路は, 非常に直線的に伸びており, 測定可能な範囲も 1000 Elo を超えている (縦軸のスケールが GNU Go のと異なる事に注意).

そこで, *s*trength speedup の物差しとして自己対戦の方を使うこととした.

本稿執筆時点で, HA8000 でのデータは採れておらず, PC クラスタを用いた 9 路の勝率が二点あるので **表 2** に示す. まだデータ数が少ないために精度が悪いので, *s*trength speedup は計算してない.

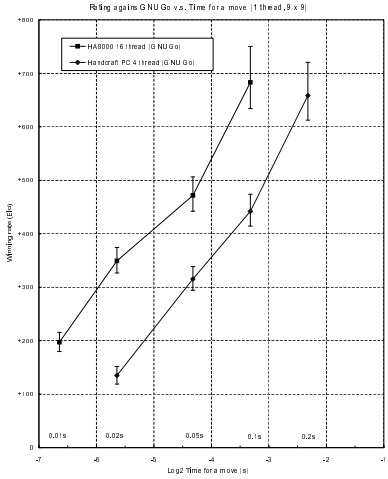

図 8 GNU Go に対する勝率 (9 路).

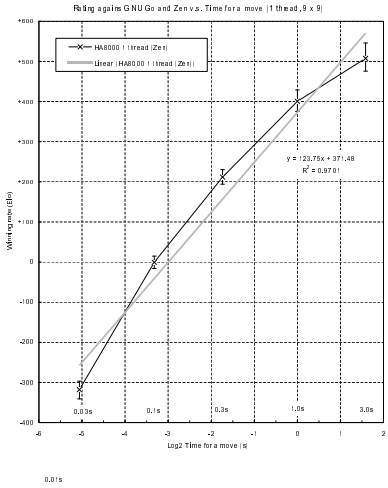

図 9 Zen に対する勝率 (9 路).

6. ま と め

Zen に粗粒度のネットワーク並列を実装し, 小規模な PC クラスタ上で性能を測定した. 大規模な HPC クラスタでの性能は, 諸般の事情でまだできてない.

並列化には, 現在のクラスタ版 MCTS 囲碁ソフトが全て採用している, 粗粒度のルート並列方式を用いたが, MPI は用いなかった. これにより, HPC クラス

タだけでなく, MPI を実装していない, あるいはできない, PC やゲームコンソールなどから成るクラスタでも動く. また, LAN だけでなく, WAN でも動かすことができるし, 運用中ノードコンピュータを動的に切り離したり繋いだりすることもできる.

ネットワーク並列の実装では, 現在広く使われている探索木共有型対称マルチスレッド並列方式に, 若干追加・修正するだけでネットワーク並列化できるよう意識し, 結果, 千行足らずの追加と修正で済んだ.

なお, 本研究には東京大学情報基盤センターの T2K オープンスーパーコンピュータ (HA8000 クラスタシステム) を使用した.

謝辞 Zen のソースコードと技術的情報を提供して下さった尾島陽児氏に深く感謝する.

参 考 文 献

1) 加藤英樹, 竹内郁雄：並列 MC/UCT アルゴリズムの実装, 第 12 回ゲーム・プログラミング ワークショップ 2007, 情報処理学会, pp.23–29 (2007). http://www.gggo.jp/.
2) Kato, H. and Takeuchi, I.: Parallel Monte-Carlo Tree Search with Simulation Servers, *Proceedings of 13th Game Programming Workshop 2008*, IPSJ (2008). http://www.gggo.jp/.
3) Cazenave, T. and Jouandeau, N.: On the Parallelization of UCT, *Proceedings of the Computer Games Workshop 2007 (CGW 2007)* (vanden Herik, J.H., Uiterwijk, J.W., Winands, M. H. and Schadd, M., eds.), MICC Technical Report Series, No. 07-06, Universiteit Maastricht, pp.93–101 (2007). http://www.ai.univ-paris8.fr/~cazenave/parallelUCT.pdf.
4) Cazenave, T. and Jouandeau, N.: A Parallel Monte-Carlo Tree Search Algorithm, van den Herik et al.[9], pp.72–80.
5) Gelly, S., Hoock, J.-B., Rimmel, A., Teytaud, O. and Kalemkarian, Y.: The Parallelization of Monte-Carlo Planning, *ICINCO* (2008). http://hal.inria.fr/docs/00/28/78/67/PDF/icin08.pdf.
6) Müller, M.: Fuego at the Computer Olympiad in Pamplona 2009: a Tournament Report, Technical report, University of Alberta, Dept. of Computing Science, TR09-09 (2009). http://www.cs.ualberta.ca/TechReports/2009/TR09-09/TR09-09.pdf.
7) Chaslot, G.M., Winands, M.H. and van den Herik, J.H.: Parallel Monte-Carlo Tree Search, vanden Herik et al.[9], pp.60–71. http://www.cs.unimaas.nl/g.chaslot/papers/parallelMCTS.pdf.
8) 東京大学情報基盤センター：HA8000 クラスタシステム (2009). http://www.cc.u-tokyo.ac.jp/service/ha8000/.
9) van den Herik, H. J., Xu, X., Ma, Z. and Winands, M. H. M.(eds.): *Computers and Games, 6th International Conference, CG 2008, Beijing, China, September 29 - October 1, 2008. Proceedings*, Lecture Notes in Computer Science, Vol.5131, Springer (2008).