TM-1308 他

# 情報処理学会　第49回　論文集

August, 1994



ICOT
Mita Kokusai Bldg. 21F
4-28 Mita 1-Chome
Minato-ku Tokyo 108 Japan
(03)3456-3191～5

Institute for New Generation Computer Technology

1T-6

# KLIC処理系核の評価

近山 隆　藤瀬 哲朗
(財) 新世代コンピュータ技術開発機構

関田 大吾　中村 豪一
(株) 三菱総合研究所

## 1 はじめに

KLIC[2]は可搬性と効率性の両立を目標に構築した、並列論理型言語KL1の並列処理系である。KLICは、KL1をCに変換してから実行可能コードを生成するという非常に可搬性の高い方式をとっている。

本稿では、その核となる逐次処理系の性能を測定し、他の類似の処理系と比較した結果を報告する。KLICはgeneric object[2]と呼ぶ柔軟な拡張機構を持ち、処理系核の変更なしに組込データ型の追加などができる。並列実装もこの機構を用いるので、逐次核の性能がそのまま通信量の少ない並列処理の各プロセサの性能を示す。

## 2 評価方法

以下の小規模なベンチマーク・プログラムについて、実行時間とコードサイズを測定し、評価した。

**nrev:** リストを反転するプログラムで、論理型言語処理系の基本性能を計るために多用されるベンチマーク[5]。30要素リストに対し10,000回繰り返した。

**qsort:** 整数データのソーティング[5]。50要素のリストに対し10,000回繰り返し実行した。

**deriv:** 記号微分プログラム[5]。times10, divide10, log10, ops8の4種のデータに対し、各々100,000回繰り返し実行した。

**primes:** 「エラトステネスのふるい」のアルゴリズムによる素数生成プログラム。10,000以下の素数をすべて生成し、個数を調べる。

**tak:** 浅い再帰呼び出しと単純な整数演算および比較を繰り返す「竹内関数」のプログラム。引数は24, 16, 8を与えた。

比較対照したのは**SICStus Prolog** vers. 2.1#8 [1], **Aquarius Prolog** vers. 1.0 [4], **JC** vers. 2.0 [3]の各システムである。このうちSICStusは、機械語を生成する方式と、抽象機械エミュレータを用いる方式の両者を比較対象とした。Aquariusは大域的な静的解析による最適化に注力した処理系で、直接機械語コードを生成する。JCはKLIC同様、Cを経由する方式である。SICStusとKLIC以外はGCをサポートしていない。

測定にはSparcStation 10/30 (SuperSparc 36MHz, 外部キャッシュなし) SunOS 4.1.3, Cコンパイラはgcc 2.5.7 `-O2 -fomit-frame-pointer`を用いた。JCでのCへのコンパイル時には、最適化オプション`-O`を指定し、算術演算を行なうプログラムでは効率向上のためにガード部分に型検査用述語を陽に書き加えた。Aquariusについては、大域的解析による最適化と、浮動小数点数を用いないことを陽に指定した。

プログラムの繰り返しには、それぞれ最速の方法、すなわちKLICでは再帰呼び出し、Prologシステムではバックトラック、JCではシステム組み込みの計測機能を用いた。KLICについては測定時間にGC時間を含み、他のシステムについては含んでいない (測定機能の不足や繰り返し方法の違いのため)。これはKLICに不利だが、KLICの方式は他の処理系に比してGCの負担が大きいことを考えると、それほどの不公平ではない。

コードサイズについては、KLICではUnixの`size`コマンドにより再配置可能オブジェクトのサイズを測定した (実行時ライブラリは含んでいない)。SICStus Prologでは、2度目のコンパイル時に報告されるコードサイズを用いている (記号表などのサイズを除くため)。

表 1: 実行時間の比較

| プログラム | K | Sf | Sc | A | J |
|---|---|---|---|---|---|
| nrev | 2,430 | 4,789 | 10,440 | 1,610 | 2,740 |
| | — | 1.97 | 4.30 | 0.66 | 1.13 |
| qsort | 3,300 | 7,320 | 14,980 | 1,920 | 3,240 |
| | | 2.22 | 4.54 | 0.58 | 0.98 |
| times10 | 2,420 | 5,659 | 12,569 | 3,090 | 3,240 |
| | — | 2.34 | 5.19 | 1.28 | 1.34 |
| divide10 | 2,900 | 5,890 | 14,390 | 3,240 | 4,370 |
| | | 2.03 | 4.96 | 1.12 | 1.51 |
| log10 | 1,060 | 3,319 | 6,299 | 1,830 | 1,670 |
| | | 3.13 | 5.94 | 1.73 | 1.58 |
| ops8 | 1,620 | 4,460 | 9,270 | 2,500 | 2,330 |
| | | 2.75 | 5.72 | 1.58 | 1.44 |
| primes | 1,600 | 2,869 | 5,349 | 2,780 | 2,170 |
| | — | 1.79 | 3.43 | 1.74 | 1.36 |
| tak | 3,020 | 6,789 | 14,820 | 1,460 | 1,590 |
| | | 1.88 | 4.09 | 0.40 | 0.44 |
| 平均 | 2,205 | 4,908 | 10,343 | 2,219 | 2,532 |
| | · | 2.23 | 4.70 | 1.01 | 1.15 |

K: KLIC; Sf: SICStus 機械語; Sc: 同 抽象機械語;
A: Aquarius; J: JC; 「平均」は幾何平均。
上段: 実行時間 (ミリ秒); 下段: KLICとの比。

## 3 評価結果

### 3.1 実行時間

表1に実行時間についての結果を掲げる。

Evaluation of KLIC System Core
Takashi CHIKAYAMA, Tetsuro FUJISE (ICOT),
Daigo SEKITA, Gouichi NAKAMURA (MRI)

**SICStusとの比較**　SICStusとの比較では、機械語コードとの比較で約2倍、抽象機械コードとで3倍から6倍程度KLICの方が速いことがわかった。

SICSのRalph Clarke Haygoodが、初期バージョンのKLICと独自の改善を施したSICStusについて、**nrev**と**qsort**のコードを詳細に比較した解析を行なった。その結果、SICStusとKLICの性能差は、KL1とPrologという言語の違いにも起因するが、システム全体の設計の違いの寄与も大きいことがわかった。SICStusではレジスタ割りつけなどの低レベル最適化まですべて自分で行なっているが、KLICはCコンパイラにまかせている。これによって生じた余力を、高レベルでの最適化に注げたことが効率向上につながったのだろう。

**nrev**プログラムではリスト連結 (append) 操作が支配的であるが、KLICでは繰り返しあたり20命令なのに対し、(独自の改善後の) SICStusでは32命令を要している。この12命令の差のうち6, 7命令程度は言語の差異によるものである (Prologではバックトラックの必要性から、単一化操作がKL1よりも複雑である)。残りの5, 6命令はシステム設計の根幹的な違いから来ており、この差を除くには大幅な再設計が必要である。

**qsort**では、ある値と比較しながらリストを分割する操作が支配的である。KLICではこの操作を要素ひとつあたり通常30命令で行なっているが、SICStusでは71命令かかる。差の41命令のうち27命令はバックトラックに備えるためで、言語の違いに起因する。残る14命令の一部も言語の差異に依存しているが、やはり言語の差に依存しない差異を完全になくすには、システムの根本的改修が必要となることがわかっている。

**Aquarius, JCとの比較**　AquariusとJCはKLICとほぼ同じ性能を示している。

両者とも**tak**についてKLICより高性能を示す。その原因はゴールスタックの実装方式の差にあると考えられる。KLICではゴールを一般データと同じヒープにおいており、GC時にも通常のデータと同様に処理している。そのため、簡単なゴールが多数生成されるプログラムではGCのオーバヘッドが大きくなる。実際、KLICではヒープを大きくすれば、3,170ミリ秒に実行時間を短縮できた。

Aquariusは整数演算を行なうプログラムの性能がKLICよりも概して良い。これは、大域的解析による冗長な手繰り操作や型検査の除去が有効なのだろうと推測できる。将来KLICにもこの手法の導入が望ましい。

一方、**primes**においては、KLICはJC, Aquariusのいずれよりも高い性能を示す。このプログラムは他と比較してメモリ消費が多く、GCのないシステムではキャッシュやTLBのヒット率の劣化が考えられる。微分プログラムにおいてもKLICの性能は比較的良い。これはKLICでは節選択にCのswitch文を用いており、Cコンパイラでの低レベル最適化がより効率的に行なわれているためであろう。

**実用規模のプログラム**　実際的な規模のプログラムの例として、KLICシステムの一部であるKL1からCへのコンパイラを測定した。KL1版はProlog版を書き変えたものだが、機能、構成、データ構造はほとんど同一である。両版のコンパイラでKL1で書いたコンパイラ自身

表 2: コードサイズの比較

| プログラム | K | Sf | /K | Sc | /K |
|---|---|---|---|---|---|
| nrev | 600 | 869 | 1.49 | 496 | 0.83 |
| qsort | 976 | 1,216 | 1.25 | 672 | 0.69 |
| deriv | 1,568 | 2,928 | 1.87 | 2,064 | 1.32 |
| primes | 1,632 | 2,352 | 1.44 | 1,424 | 0.87 |
| tak | 544 | 608 | 1.12 | 288 | 0.53 |
| 平均 | 960 | 1,355 | 1.41 | 776 | 0.81 |

K: KLIC; Sf: SICStus 機械語; Sc: 同 抽象機械語。
単位はバイト、平均は幾何平均。

(約5,000行) をCコード (約50,000行) に変換するのにKLICは20.1秒、SICStus (機械語版) では45.1秒を要した (cshのtimeコマンドで計測)。[1]大規模プログラムでも2倍以上の速度差が確認できたわけである。

### 3.2　コードサイズ

表2にオブジェクトコードサイズを示す。SICStus以外のシステムでは対象プログラムのみのサイズを測定することが困難であるため比較の対象にしていない。

表に示すように、SICStusの機械語コードはKLICより40%強大きく、抽象機械語コードでも20%減る程度にすぎない。これはKLICでコンパイルしたコードと実行時システムとの機能分割が巧妙であることを示す。

### 3.3　コンパイル時間

長いコンパイル時間はKLICの欠点である。SICStusが各プログラムを100ミリ秒以内で処理するのに対し、KLICは数秒を要する。JCもほぼ同様で、両者ともCコンパイラの時間が支配的である。[2]このためKLICでは、極力再コンパイルを行なわない仕組み (makeと同様の機構、再コンパイルが不要なデバガ) を備えた。

## 4　おわりに

KLIC処理系核部分の評価を行ない、他の類似処理系と比較した。KLICはベンチマークプログラム、さらには実際的なサイズのプログラムにおいても高い性能を示すことがわかり、基本的な設計部分での優位性が判明した。一方、今後、大域的な静的解析を行なうことにより型検査、手繰り操作などのオーバヘッドを減らし、さらに性能を向上させる余地があることがわかった。

## 参考文献


[1] M. Carlsson *et al.* : *SICStus Prolog User's Manual*, 1993.

[2] T. Chikayama *et al.* : *A Portable and Efficient Implementation of KL1*, PLILP'94, 1994, *to appear*.

[3] D. Gudeman *et al.* : *User Manual for* jc　*A janus compiler, version 2.0*, 1994.

[4] R. C. Haygood: *Aquarius Prolog User Manual*, 1993.

[5] D. H. D. Warren: *Implementing Prolog - compiling predicate logic programs*. Research rep. 39&40, Dept. of Artificial Intelligence, Univ. of Edinburgh, 1977.


[1]分割コンパイルできないAquariusやJCでは処理できなかった。
[2]Aquariusは**nrev**に対してでさえ20秒程度かかる。