Calcul numeric în Java

Lect. Bogdan OANCEA
obogdan@xnet.ro

Java is recognized as an excellent object oriented programming language with support for network applications, graphics and distributed applications. For numerical intensive applications, Java is still behind languages like FORTRAN or C. This paper presents the implementation and performance of a Java package - JLA (Java for Linear Algebra) . JLA implements a subset of the standard subroutines for matrix factorizations and linear system solving from BLAS and LAPACK packages.
Keywords: *Java, numerical computing, linear algebra.*

Introducere

Modelele matematice sunt în prezent utilizate în cele mai diverse domenii de activitate. Dezvoltarea si cresterea performantelor tehnicii de calcul a condus la aparitia unor sisteme informatice destinate rezolvarii numerice a modelelor matematice. Utilizate la început aproape exclusiv în domeniul militar, modelele matematice s-au extins rapid astfel încât azi la întâlnim în cele mai diverse domenii: industria petroliera, industria chimica, în probleme de transport si distributie, industria aviatiei etc.

Algoritmii de factorizare a matricelor si rezolvare a sistemelor liniare sunt bine cunoscuti si în prezent exista pachete de programe performante ce implementeaza acesti algoritmi.

În ultimii ani au aparut noi limbaje de programare, unul dintre acestea fiind si limbajul Java. Mai mult decât orice alt limbaj nou aparut, Java se dovedeste limbajul de programare cu cel mai mare potential de a deveni un excelent mediu pentru dezvoltarea aplicatiilor de mari dimensiuni ce necesita calcule numerice intensive.

Implementarile curente ale masinii virtuale Java si a limbajului de programare Java suporta în mod direct numai masive unidimensionale. Java simuleaza masivele bidimensionale (matricele) prin masive unidimensionale ale caror elemente sunt ele însele masive unidimensionale ("vectori de vectori"). Aceasta abordare furnizeaza programatorilor o mare flexibilitate în scrierea programelor, dar pentru compilatoare acest lucru reprezinta un dezavantaj major întrucât nu se pot folosi tehnici avansate de optimizare a codului. Dificultatile provin din faptul ca masivele îsi pot schimba forma (dimensiunea) si mai multe elemente dintr-un masiv - elemente ce sunt de fapt pointeri - pot indica aceeasi locatie de memorie. Alt dezavantaj major provine din verificarile pentru exceptiile de depasire a limitelor dimensiunii masivului. Se cunoaste faptul ca în limbajul Java, indicii sunt verificati pentru exceptia de depasire a dimensiunii maxime a masivului iar pointerii sunt verificati pentru exceptia de atribuire a valorii NULL. Accesul la un element A[i][j] implica doua verificari pentru pointeri, A si A[i], si doua verificari ale indicilor, i si j. Aceste verificari repetate pentru detectarea exceptiilor vor afecta în mod negativ performantele programelor de calcul numeric.

Rezultate experimentale

Pentru experimente am utilizat un calculator cu procesor AMD K6-2 / 350 MHz, cu 64 Mb memorie principala si Java 2 SDK sub sistemul de operare Linux. În tabelul 1 sunt prezentate performantele rutinelor JLA pentru înmultirea matricelor (DGEMM), comparativ cu performantele obtinute cu o implementare standard BLAS în limbajul C (CBLAS) si cu o implementare Java ce utilizeaza "vectori de vectori" (denumite în continuare masive Java) pentru reprezentarea matricelor. Performantele ru-

tinelor mentionate sunt exprimate în MFLOPS (milioane de operatii în virgula mobila pe secunda).

Tabelul 1. Performantele rutinei DGEMM exprimate în MFLOPS

Dimensiunea matricei	CBLAS	JLA	masive Java
200	24.20	10.51	4.18
300	22.01	11.01	4.23
400	23.03	11.83	4.67
500	23.00	12.92	5.01
600	23.51	13.44	5.18
700	23.85	14.15	5.2
800	23.61	14.00	5.09
900	23.87	15.80	4.82
1000	23.40	13.50	4.62
1200	23.48	13.40	4.03
1500	22.67	12.00	4.00

Pachetul JLA implementeaza si rutina DGEMM ce lucreaza la nivel de bloc. Utilizarea variantei la nivel de bloc permite obtinerea unor performante superioare prin stabilirea dimensiunii blocului astfel încât datele aduse în memoria cache sa fie reutilizate cât mai mult posibil pentru calcule înainte ca memoria cache sa fie evacuata si sa fie încarcat un nou bloc de matrice. Rezultatele rutinei la nivel de bloc sunt prezentate în tabelul 2.

Tabelul 2. Performantele rutinei DGEMM la nivel de bloc exprimate în MFLOPS

Dimensiunea blocului	Dimensiunea matricei				
	600	800	1000	1200	1400
1	13.50	14.05	13.73	13.68	13.55
2	14.60	15.05	15.20	15.06	14.87
4	15.25	15.83	15.82	15.64	15.50
8	15.33	15.73	15.65	15.75	15.84
16	15.00	15.60	15.86	15.92	16.00
32	14.95	15.60	15.86	15.85	15.90
64	14.80	15.40	15.70	15.80	15.60

În figura 1 sunt reprezentate grafic performantele rutinei DGEMM prezentate în tabelul 1. Se poate observa usor ca rutina DGEMM din pachetul JLA obtine pâna la 66% din performanta atinsa de implementarea CBLAS în timp ce rutina care lucreaza cu masive bidimensionale reprezentate ca "vectori de vectori" (masive Java) nu obtine decât aproximativ 20% din performantele CBLAS.

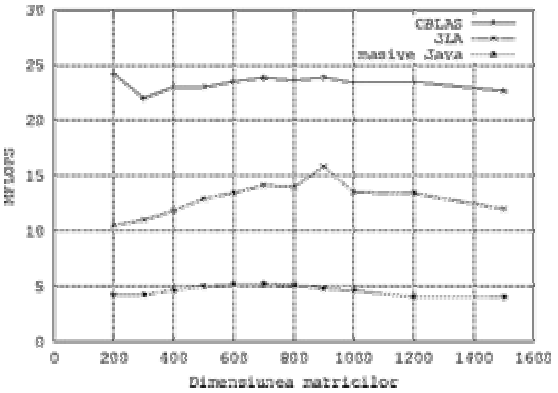

Fig. 1. Performantele rutinei DGEMM pentru implementarea CBLAS, JLA si o implementare utilizând masive bidimensionale Java

Utilizarea algoritmilor la nivel de bloc permite o îmbunatatire a performantelor prin reutilizarea blocurilor de matrice aduse în memoria cache. Totusi, pentru pachetul JLA, aceasta crestere a performantei nu este atât de mare ca în cazul rutinelor scrise în limbajul C sau FORTRAN. În figura 2 sunt reprezentate grafic rezultatele prezentate în tabelul 2. Se observa ca, pentru sistemul de calcul utilizat, dimensiunea optima a blocului de matrice este de 16. Pentru aceasta dimensiune se obtine o îmbunatatire a performantelor cu aproximativ 15% fata de cazul algoritmilor ce nu lucreaza la nivel de bloc.

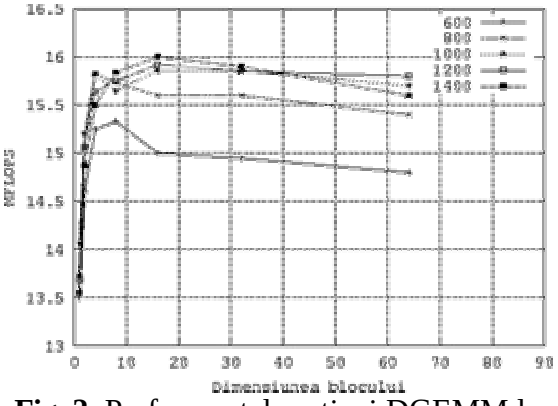

Fig. 2. Performantele rutinei DGEMM la nivel de bloc a pachetului JLA pentru diferite dimensiunii ale matricei.

În tabelul 3 sunt prezentate performantele obtinute de rutina de factorizare LU din pachetul JLA, comparativ cu o implementare a factorizarii LU utilizând masive bi-

dimensionale Java ("vectori de vectori") iar în figura 3 aceste performante sunt reprezentate grafic.

Tabelul 3. Performantele rutinei de factorizare LU exprimate în MFLOPS

Dimensiunea matricei	LU - JLA	LU - masive Java
100	3.83	0.89
200	10.86	3.63
300	13.65	3.85
400	14.60	4.21
500	15.00	4.82
600	15.00	4.79
700	15.25	4.92
800	14.95	5.01
900	15.24	5.21
1000	14.83	4.83
1100	15.16	4.77
1200	14.67	4.56
1300	15.14	4.03
1400	14.55	4.21
1500	15.10	3.83

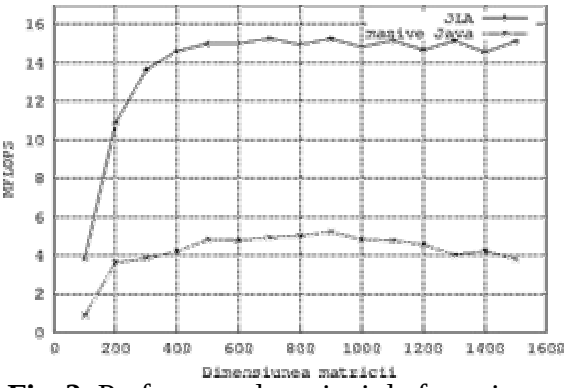

Fig. 3. Performantele rutinei de factorizare LU a pachetului JLA si pentru o implementare utilizând masive bidimensionale Java

Din tabelul 3 se poate observa foarte usor ca factorizarea LU implementata cu masive bidimensionale Java nu obtine decât aproximativ 30% din performantele rutinei de factorizare LU din pachetul JLA. Rezultatele obtinute cu rutina de factorizare LU la nivel de bloc sunt prezentate în tabelul 4 si figura 4.

Tabelul 4. Performantele rutinei de factorizare LU la nivel de bloc exprimate în MFLOPS

Dimensiunea blocului	Dimensiunea matricei						
	200	400	600	800	1000	1200	1400
2	6.20	12.80	14.60	15.05	15.20	15.06	14.87
4	6.33	13.48	15.45	15.83	15.82	15.64	15.50
8	6.43	12.90	15.33	15.73	15.65	15.75	15.84
16	5.22	12.85	15.00	15.60	15.86	15.92	16.00
32	5.43	12.70	14.95	15.60	15.86	15.85	15.90
64	5.57	12.65	14.80	15.40	15.70	15.80	15.66

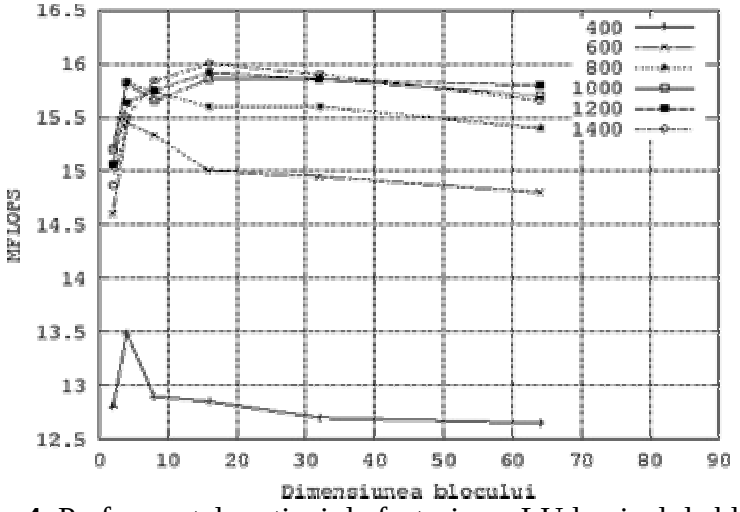

Fig. 4. Performantele rutinei de factorizare LU la nivel de bloc a pachetului JLA pentru diferite dimensiunii ale matricii

La fel ca si la rutina de înmultire a matricelor DGEMM, se observa ca în cazul limbajului Java, utilizarea unor algoritmi la nivel de bloc aduce o îmbunatatire a performantelor, dar aceasta este de doar 10% în cazul cel mai bun. Aceleasi concluzii se pot desprinde si în ceea ce priveste factorizarea Cholesky. Rezultatele obtinute pentru rutina de factorizare Cholesky la nivel de bloc sunt prezentate în tabelul 5 si în figura 5.

Tabelul 5. Performantele rutinei de factorizare Cholesky la nivel de bloc

Dimensiunea blocului	Dimensiunea matricei						
	200	400	600	800	1000	1200	1400
2	3.32	9.68	12.45	13.80	14.47	14.33	15.22
4	3.34	9.85	12.77	13.92	14.66	14.92	15.42
8	3.40	8.57	12.32	13.96	14.32	15.10	15.25
16	2.28	8.37	12.94	14.10	14.40	14.92	15.23
32	2.22	8.18	12.01	13.62	14.35	14.85	15.15
64	2.28	8.50	11.96	13.52	14.26	14.72	14.95

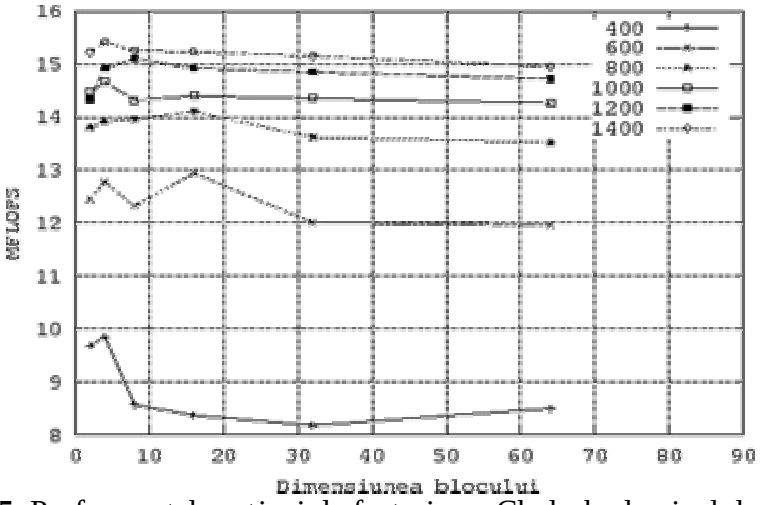

Fig. 5. Performantele rutinei de factorizare Cholesky la nivel de bloc a pachetului JLA pentru diferite dimensiunii ale matricei

În final prezentam rezultatele obtinute pentru rutinele de rezolvare a sistemelor liniare generale, atât pentru implementarea JLA cât si pentru o implementare utilizând masive bidimensionale Java - tabelul 6 si figura 6.

Tabelul 6. Performantele rutinei de rezolvare a sistemelor liniare exprimate în MFLOPS

Dimensiunea matricei	JLA	masive Java
200	6.50	2.01
400	12.27	3.98
600	14.42	4.69
800	14.60	5.11
1000	14.62	4.93
1200	14.53	4.86
1400	14.45	4.11

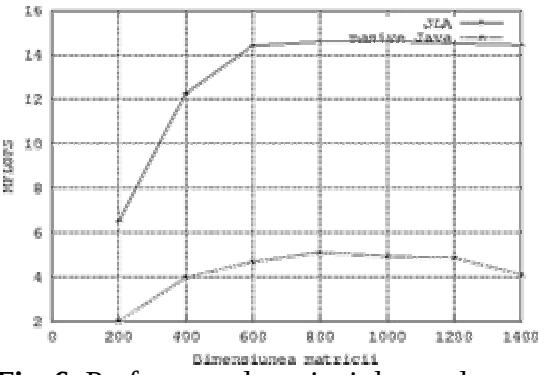

Fig. 6. Performantele rutinei de rezolvare a sistemelor liniare a pachetului JLA si pentru o implementare utilizând masive bidimensionale Java

Concluzii

Pe ansamblu, rutinele JLA obtin în jur de 55 - 75 % din performantele rutinelor echivalente BLAS sau LAPACK. Aceste rezultate au fost posibile prin tehnici de

optimizare a codului, astfel încât sa se obtina o reutilizare a memoriei cache, reducerea operatiilor auxiliare prin deplasarea invariantilor si prin utilizarea tehnicii de desfasurare a buclelor de program (loop-unrolling). Performantele obtinute de rutinele pachetului JLA în rezolvarea sistemelor liniare ne îndreptatesc sa credem ca utilizând un suport adecvat din partea compilatoarelor, limbajul de programare Java poate deveni un mediu ideal de dezvoltare a programelor de calcul stiintific, incluzând aplicatii de mari dimensiuni din domeniul economic, precum sunt cele prezentate în introducere.

Bibliografie

1. J. Armstrong, R. Black, D. Laxton, and D. Rose. The Bank of Canada'"s New Quarterly Projection Model QPM. Part 2: A Robust Method for Simulating Forward-Looking Models. Technical Report No 73, Otawa: The Bank of Canada, February 1995.

2. F. Brayton and P. Tinsley. A guide to FRB/US : A Macroeconomic Model of the United States. Technical Report Finance an Economics Discussion Series, Federeal Reserve Board, 1996.

3. C.L. Lawson, R.J. Hanson, D.R. Kincaid, and F.T. Krogh. Basic linear algebra subprograms for Fortran usage. *ACM Trans. Math. Soft.*, (5(3):308-323), 1979.

4. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.*, (14(1):1-17), March 1988.

5. J. Dongarra, J. Du Croz, S. Hammarling, and I. Duff. A set of level 3 basic linear Algebra subprograms. *ACM Trans. Math. Soft.*, (16(1):1-17), 1990.

6. E. Anderson, Z. Bai, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, S. Ostrouchov, and D. Sorensen. *LAPACK Users's Guide*. SIAM, Philadelphia, 1992.