Méthodes de simulation au niveau transfert de registres de processeurs et machines de traitement du signal

Yves SOREL

INRIA - Domaine de Voluceau - BP 105 - 78153 Rocquencourt - France

RÉSUME

Les machines cibles considérées seront construites avec des processeurs de traitement du signal. Dans une première partie on étudie les particularités des architectures internes des processeurs, puis les particularités des architectures des machines elles même.Ceci permet de réaliser une description représentative.Les processeurs sont décrits au niveau transfert de registres par un graphe flot de données.On étudie l'efficacité de tels simulateurs afin de les exécuter dans des temps raisonnables. Pour cela on introduit des méthodes de mesures de performances.

SUMMARY

Ours signal processing machines are built with signal processing processors. First we study internal architecture features of such processors, then machines architecture features. This allows a representative description. Processors are described at the register transfert level using a data flow graph. We study the efficiency of such simulators in order to execute them quikly using performance analysis.

1. CARACTERISTIQUES DES ARCHITECTURES

Un simulateur de machine offre des avantages considérables. Il permet de faire la mise au point logique des programmes, ainsi que l'évaluation des performances d'une machine sans avoir à l'acheter ou à la construire.
Plusieurs langages dits HDL (pour Hardware description language) ont été conçus pour décrire une machine informatique au niveau transfert de registres (RTL) <BARBACCI81>, <SOREL83>.
Ces langages, comme ISPS <BARBACCI81>, permettent une description aisée de la machine choisie, cependant les temps d'exécution d'un tel simulateur deviennent prohibitifs dès que le programme de description est très détaillé, de plus le paralléllisme n'est pas parfaitement pris en compte <SOREL85>.
Le langage OCCAM conçu pour décrire des applications concurrentes a été utilisé, afin de rendre compte de tous les parallélismes possibles (étudiés dans le paragraphe suivant) nécessaires à la réalisation d'un simulateur efficace.
L'autre avantage très important qu'offre ce langage, est qu'il est directement conçu pour permettre l'implantation des programmes OCCAM <OCCAM88> sur un réseau de processeurs de type TRANSPUTEUR <TRANS88>. On peut ainsi implanter un simulateur de machine multiprocesseur sur une machine cible multiprocesseur.
L'application qui nous intéresse, en l'occurrence la simulation d'une machine multi-TMS 32020, sera donc implantée sur un réseau de transputers à l'aide d'OCCAM.

1.1. Les processeurs

Nous étudions dans ce qui suit les différentes techniques qui permettent de prendre en compte le parallélisme interne au processeur.
On dégage deux types de parallélisme <POLY87> :
- un parallélisme au niveau d'une séquence d'instructions (programme) ou pipe-line appelé parallélisme vertical. Les différentes étapes du pipe-line doivent être sensiblement de même durée, afin que le pipe-line commence et termine le traitement d'une instruction à intervalles réguliers (condition requise pour les processeurs de traitement du signal du fait de la synchronisation des algorithmes implantés).
- un parallélisme au niveau d'une instruction, qui permet, dans le cadre d'une même instruction, de faire appel simultanément à différentes ressources. Ce parallélisme est appelé parallélisme horizontal.
Les processeurs considérés ont une architecture de type Havard, caractérisée par la séparation de l'espace mémoire donnée et de l'espace mémoire programme. La mémoire est une ressource fortement sollicitée par le processeur. Elle l'est d'autant plus que ce dernier est pipeliné ; son découpage en deux blocs indépendants va lui permettre de satisfaire deux requêtes simultanément. Cette séparation implique l'implantation de deux bus : un bus de donnée et un bus programme (exemple du TMS32020).
Avant de pouvoir exécuter une instruction, le CPU du microprocesseur cherche le code instruction correspondant, étape FETCH, puis, il détermine les tâches à faire étape DECOD et les exécute au fur et à mesure, étape EXEC. Les étapes FETCH-DECOD et l'étape EXEC peuvent être pipelinées.

L'introduction d'autres niveaux de pipe-line pose des problèmes. En effet la profondeur du pipe-line est liée à la séquence d'instructions à exécuter. Dans le cas d'un processeur réel cette profondeur est fixée, cela conduit souvent à introduire dans le programme assembleur des instructions ne faisant rien pour respecter un fonctionnement correct. Dans le cas d'un simulateur on peut envisager un pipe-line reconfigurable qui s'adapte au programme. Dans tous les cas une analyse préalable du programme qui détecte les dépendances entre instructions est nécessaire.
Ces processeurs possèdent généralement plusieurs unités de calcul séparées (unité de calcul arithmétique ,multiplieur cablé, unité de calcul d'adresse). Ceci permet d'accéder en parallèle à plusieurs unités dans une même instruction, à condition d'utiliser des chemins de données différents.

1.2. Les machines

Elles sont construites à l'aide des processeurs du type de ceux décrits plus haut. On s'interresse à des machines de type MIMD (multiple instructions, multiple datas) ou machines multiprocesseur <FLYNN66>. Dans ce cas les processeurs connectés en réseau peuvent communiquer de deux manières différentes :
-par l'intermédiaire d'une mémoire partagée
-par passage de messages.
Dans les deux cas les moyens de communication sont soit des connexions point à point (informations passées en parallèle ou en série) soit multipoint (utilisation de bus). Avec ces moyens de connexion on réalise des réseaux de processeurs qui peuvent être complètement interconnectés, en anneau, en étoile, en matrice, en hypercube, en échange- permutation etc ou bien mélangeant ces différentes techniques <Wu84>.
Les machines que nous simulons peuvent être d'un de ces types, l'interconnexion ne pose pas de difficulté, c'est l'écriture des noyaux temps réel exécutant les protocoles de communication particuliers qui peut être complexes à mettre en oeuvre.

2. SIMULATION

2.1. Description des processeurs

Le simulateur doit avoir un comportement vu de l'extérieur conforme au comportement réel du processeur. La façon dont on simule le comportement du processeur peut être liée directement à l'architecture réelle ou être améliorée en s'inspirant de nouvelles techniques plus performantes expérimentées dans d'autres processeurs de génération plus récente. Ceci permet de faire évoluer le simulateur, par exemple, le jeu d'instruction peut être symétrisé (chaque opération possède son symétrique : ex. load et store) ou orthogonalisé (indépendance entre les codes opérations et les modes d'adressages). Ceci est un des objectifs des processeurs RISC (Reduced Instruction Set Computer) <PATT85> .
On peut rajouter une mémoire cache basée sur la propriété de localité que possèdent le déroulement des instructions et l'accès aux données <CHAUD86>.

Principes

Les processeurs sont décrits au niveau transfert de regitres. On utilise comme point de départ le schéma bloc donné par le constructeur. Il met en évidence les registres, les mémoires internes et les chemins de données directs ou par l'intermédiaire de bus internes, qui relient les registres entre eux. Ces chemins traversent éventuellement une unité de calcul.
On construit un graphe flot de données dont les arcs sont les chemins de données (et d'instruction) et les noeuds des registres avec l'unité de calcul associée.
Le problème des cycles dans le graphe est résolu, car tout cycle comprend au moins une mémoire (registre) qui joue le rôle de retard coupant le cycle considéré. En effet pour chaque instruction, on exécute le graphe flot de données complètement. On utilise ce qu'il y a dans les registres à l'intant n, puis on les remet à jour. L'instruction suivante à l'instant suivant n+1 (autre exécution du graphe flot de données) utilisera les valeurs des registres calculés précédemment. Le retard permet d'éviter l'indéterminisme qui pourrait être lié à l'arrivée simultanée de plusieurs entrées, dont certaines seraient des rebouclages.
En fonction de l'instruction en cours d'exécution, certains noeuds et arcs du graphe ne seront pas utilisés. L'étude du jeu d'instructions va déterminer pour chaque instruction lesquels des noeuds seront activés. Chaque noeud peut recevoir des arcs qui lui indiquent s'il a à s'exécuter ou non. Ces informations de non activité sont fournies par les noeuds précédents.
Ce graphe doit mettre en évidence les deux types de parallélisme vus plus haut. Le pipe-line correspond à des noeuds mis en série le parallélisme horizontal par des noeud mis en parallèle. Nous obtenons ainsi des graphes à structure série parallèle dont les cycles (rebouclages) sont ignorés, ils correspondent à des noeuds qui sont soit uniquement producteur, soit uniquement consommateur de données.

Exemple de description le TMS 32020

Le processeurs TMS 32020 de Texas instrument <TEXAS86> l'un des plus utilisé dans les applications de traitement du signal a été choisi pour décrire la méthode.
Pour simplifier les schémas on se place dans le cas des mémoires de programme et de données internes. A partir du schéma bloc fig1. on définit le graphe flot de données complet fig2.

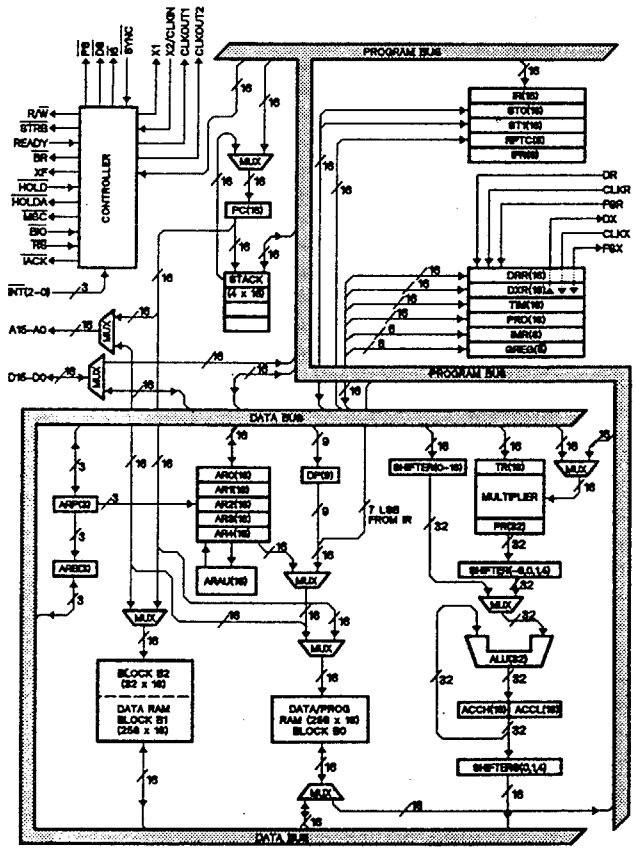

Fig1. Architecture interne du TMS32020

Description des noeuds du graphe :
PMEM : mémoire programme, contient les codes instruction prétraités et les données pour le multiplieur (par ex. coefficients d'un filtre transversal),
DMEM : mémoire données, produit l'opérande pour les unités de calcul MUL et ALU, reçoit le résultat de ALU, produit et reçoit des données de l'extérieur du processeur par le noeud IO,

FETCH : va chercher l'instruction suivante n+1 et lance en pipe line l'exécution de l'instruction n après réception d'un signal de fin d'instruction, envoie le type d'opération à effectuer,
EAD : calcule l'adresse effective dans le cas de l'adressage direct en ajoutant le contenu du pointeur de page DP à l'adresse contenue dans l'instruction, dans le cas de l'adressage indirect utilise comme adresse effective la valeur d'un des cinq registres AR pointé par le registre ARP,
ARAU : modifie le registre AR courant (incrémentation, décrémentation),
ALU : effectue les calculs arithmétiques et logiques,
MUL : effectue la multiplication du registre T ou d'un élément de mémoire programme par un opérande venant de DMEM (adressage direct ou indirect) ou de l'instruction (adressage immédiat),
IO : effectue l'interface avec l'extérieur du processeur, entrées sorties parallèles et séries, interruptions, signaux de contrôle.

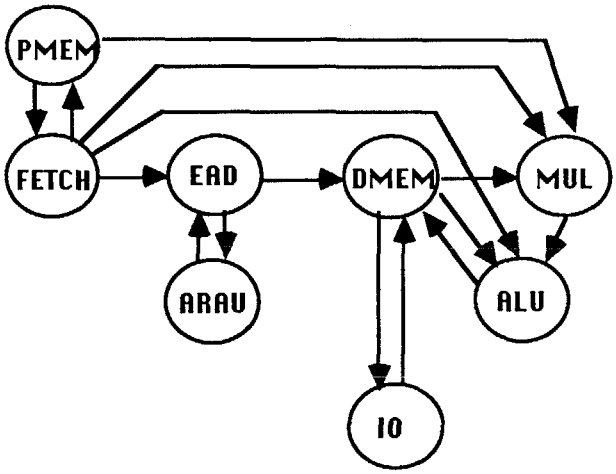

Fig2. Graphe flot de données

Pour ce processeur on peut effectuer en pipe-line la phase de recherche de l'instruction FETCH et la phase d'exécution EXEC. L' exécution et le décodage étant mêlés, toutes les opérations de décodage qui peuvent être effectuées d'une manière statique sont réalisées dans un traitement préliminaire. On pré-calcule le décodage à partir du programme assembleur source. Ceci permet d'écourter la phase de décodage pendant l'exécution.
Dans la phase EXEC on peut effectuer en parallèle les modifications sur les registres pointeurs de mémoire données et les calculs (calcul arithmétiques et logique faisant intervenir l'unité arithmétique et logique ALU et le multiplieur MUL). En effet dès que l'adresse de l'opérande en cours d'exécution a été calculée, AR peut être incrémenté ou décrémenté et le registre ARP qui pointe sur un des cinq registres AR peut être modifié en même temps que les calculs s'effectuent sur ce même opérande.
Examinons ce qui se passe pour quelques instructions du TMS32020.
Tout d'abord une instruction simple,
ADD *+,2,3. Si on suppose que le pointeur de registres d'adresses ARP vaut 0 (premier des cinq registres AR), elle s'exécutera de la manière suivante :
-recherche de l'instruction dans la mémoire programme et incrémentation du pointeur de programme PC,
-addition à l'accumulateur de l'opérande shifté de 2, son adresse a été calculée à partir du pointeur courant AR pointé par ARP (adressage indirect), incrémentation de AR, modification du pointeur ARP, il pointe maintenant sur le troisième registre AR.

Ceci correspond pour le graphe aux actions suivantes :
Durant la phase FETCH l'instruction est recherchée par le noeud FETCH. Il fournit l'adresse courante à la mémoire programme PMEM et récupère l'instruction, puis incrémente le pointeur de programme pour l'instruction suivante. Le noeud FETCH à partir de l'instruction éventuellement prédécodée, active le noeud EAD (calcul de l'adresse effective adressage indirect et direct) en envoyant une information indiquant la nature de l'opération à effectuer. Cette information : typop (dans ce cas une addition) est concaténée à l'adresse qu'il fournit à la mémoire de donnée, noeud DMEM. Ce dernier envoie l'opérande concaténé à typop au noeud ALU. Il envoie aussi aux noeuds MUL et IO une information de non activité. ALU effectue un shift de l'opérande puis l'additionne avec l'accumulateur et met le résultat dans l'accumulateur. Dès que l'opération est terminée ALU envoie un signal de fin.
Quelque soit le type d'adressage indirect comme ci-dessus, direct ou immédiat l'opérande parvient aux unités de calcul accompagné du type d'opération à exécuter typop.
Dès que l'adresse de l'opérande a été envoyée par AR vers DMEM, le noeud EAD active le noeud ARAU en lui envoyant une information typop (incrémentation, décrémentation du registre AR courant ou dans le cas de l'adressage indirect indexé, mise à jour du registre AR courant par la somme du contenu de ce registre avec le registre AR(0)), ARP est ensuite modifié. Ceci s'effectue en parallèle avec les opérations précédentes (accès à DMEM, ALU).

Examinons une instruction plus complexe,
MACD pma,*+,3.
Cette instruction multiplie un élément de la mémoire données (il peut être adressé indirectement par AR comme dans l'exemple ou directement) par un élément de la mémoire programme, puis ajoute le produit obtenu précédemment (ici shifté de 3) avec l'accumulateur. L'élément de la mémoire données est recopié dans l'élément suivant (décalage du retard).
Cette instruction permet de réaliser en un coup l'un des k étages d'un produit de convolution de la forme
$yn = \sum hi*x(n-i)$ i variant de 0 à k.
Cette instruction doit être répétée k fois pour obtenir le produit de convolution complet. Pour cela on fait précéder une instruction MACD par une instruction particulière de répétition RPT k. La mémoire programme contient les coefficients hi pointés par le compteur de programme incrémenté à chaque instruction et la mémoire données contient les entrées retardées x(n-i) pointées par AR incrémenté à chaque instruction.
Dans le cas où l'on utilise RPT, l'accumulation s'effectue en pipe-line avec la multiplication pendant la durée des k instructions.
Ceci correspond pour le graphe aux actions suivantes :
Le début est identique à l'exemple précédent. DMEM fournit à MUL un de ses opérande, l'autre est fournit par PMEM. MUL et ALU s'exécutent en pipe-line, c'est à dire que ALU accumule le produit précédent alors que MUL calcule le produit courant. MUL et ALU envoient un signal de fin lorsqu'ils ont terminé, éventuellement après k utilisations si on a mis l'instruction de répétition qui positionne un compteur de répétition à k.

Le parcourt du graphe flot de données est déterminé par la règle d'activation suivante :
Chaque noeud attend une ou plusieurs entrées, dès qu'il en reçoit une, il exécute sa partie calcul et transfert de registres, puis produit un ou plusieurs résultats et une ou plusieurs informations de non activité aux noeuds auxquel il est connecté.
Les noeuds terminaux envoient tous une information de fin au noeud FETCH qui peut alors réarmer son pipe-line pour l'instruction suivante (pour plus de clarté les arcs correspondant à ces signaux de fin n'apparaissent pas dans le graphe de la fig2.).

2.2. Description des machines

les processeurs décrits comme ci dessus, sont instanciés autant de fois que nécessaire puis connectés soit directement, soit par l'intermédiaire d'un processus qui

joue le rôle de bus physique incluant un mécanisme d'arbitrage, soit à un processus mémoire partagée contenant éventuellement aussi un mécanisme d'arbitrage. On donne en exemple le schéma d'une machine à deux processeurs TMS32020 communicant par une mémoire globale partagée fig3. Un mécanisme d'arbitrage (par ex. un mécanisme équitable de round-robin) implanté dans la mémoire globale indique lequel des processeurs, ayant fait une demande d'accès, est autorisé à lire ou à écrire dans la mémoire. Les demandes d'accès se font avec les sorties BR des processeurs l'autorisation est envoyée par la mémoire sur les entrées READY. Les sorties A15-A0 produisent l'adresse de l'élément de mémoire lu ou écrit sur D15-D0.

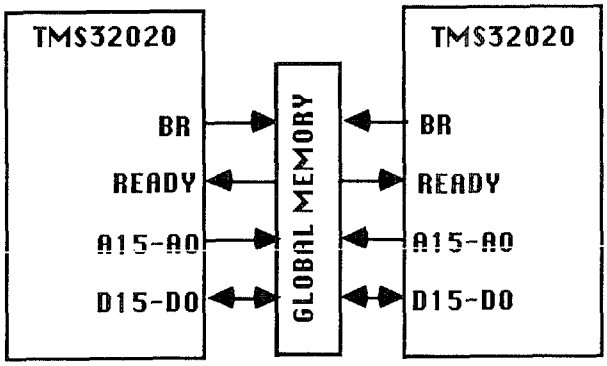

Fig3. Machine à deux TMS32020

2.3. Génération des simulateurs

Un processeur est un ensemble de processus OCCAM mis en parallèle. Chaque noeud du graphe flot de données est représenté par un processus. Les processus communiquent par rendez-vous à l'aide des canaux OCCAM qui représentent les arcs du graphe.
La machine est un programme OCCAM dans lequel on a mis en parallèle les processus correspondant aux processeurs (ils ont été instanciés autant de fois que nécessaire) et les processus particuliers représentant les bus et les mémoires externes. Les communications ont aussi lieu à travers des canaux. Dans l'exemple de la fig3. on a instancié deux fois le simulateur du processeur TMS32020 et on les a connectés à un processus mémoire globale.
Le programme après compilation est réparti sur un réseau de transputers. Le simulateur exécute les programmes en assembleur qui ont été chargés dans les mémoires programme des différents processeurs. Ceux ci font appel à un noyau temps réel dont les sous-programmes gèrent les mécanismes de synchronisation nécessaires au bon déroulement des communications entre processeurs, entre processeurs et bus, entre processeurs et mémoire partagée.

3. Evaluation des performances

Les simulateurs décrits précédemment ayant une grande complexité doivent être accompagnés d'outils de visualisation, permettant d'évaluer leurs performances et effectuer la mise au point des programmes assembleurs qu'ils exécutent. Ces outils constitueront un moniteur de mise au point pour le simulateur.
Le premier outil nécessaire pour assurer le confort de l'utilisateur est de relier le simulateur à des entrées sorties fichier. Les mémoires programmes et données des processeurs sont ainsi accessibles plus facilement. On doit pouvoir lire et écrire dans des fichiers les entrées sorties des processeurs (noeud IO). On souhaite aussi pouvoir stoker en mémoire l'historique de certaines variables internes importantes pour étudier le comportement du simulateur.
Ceci est réalisable car une interface matérielle et logicielle existe entre le ou les transputers qui exécutent le simulateur et un système de gestion de fichiers, en l'occurence MSDOS puisque nous avons utilisé une carte à base de transputers au format IBMPC.
De la même manière, on peut lire le clavier ou écrire sur l'écran du PC. C'est l'attente de l'envoie d'un caractère du clavier qui déclanchera la mise en fonction des différent modes du moniteur de mise au point. Le noeud FETCH connait la terminaison de chaque instruction et on sait à ce momment mettre en attente tous les processus du simulateur. On pourra donc les interroger chacun son tour, pour lire les valeurs des variables internes et éventuellement les modifier. Ceci permet le pas à pas et l'arrêt sur instruction . Un programme écrit en C sur le PC effectue l'affichage des informations nécessaires (nom des commandes en cours et résultats produits)
Il est évident que le problème se complique énormément lorsque le simulateur est réparti sur un réseau de transputeurs.

4. Résultats

Quelques essais de description de processeurs de signaux (NEC 7720, TMS 32020 par exemple) en ISPS et en OCCAM ont montré la supériorité de ce dernier langage du fait :
-d'une notion plus claire de parallélisme,
-de la disponibilité de machines cibles parfaitement adaptées avec les multitransputers qui ont évolué au cours de l'étude jusqu'au T800 à 20 Mhz actuel et qui permettent des vitesses plus grandes.
Par exemple, simulant un modem 4800bps sur un TMS32020, on est arrivé aux performances suivantes en instructions du TMS32020/seconde :
-avec ISPS : 28 ins/s sur PDP10, 20 ins/s sur VAX 11/750
-avec OCCAM : 33 ins/s sur un T414 20 mhz. Le simulateur OCCAM permettant de décrire des montages multi 32020 contrairement à celui en ISPS (temps de simulation trop long sur des machines cibles séquentielles).

La méthode proposée a plusieurs intérêts. D'une part elle permet une description naturelle du processeur, on traduit directement le schéma bloc du constructeur et le jeu d'instructions. D'autre part grâce à OCCAM et au transputer, elle permet une prise en compte simple du parallélisme interne des processeurs ainsi que celui des machines. De plus on a la possibilité d'exécuter les simulateurs sur des machines cibles multiprocesseur afin d'en améliorer les performances.

BIBLIOGRAPHIE

<CHAUD86>G.M.CHAUDHRY,J.S.BEDI.Cache interleaving in multiprocessor systems.Microprocessing and Microprogramming 18 (1986) 205-214.
<FLYNN66> Flynn M.J. Very high speed computing system. proc. of the IEEE, Vol 54, Dec 1966, pp 1901-1907.
<PATT85> Patterson D.A. Reduced Instruction Set Computers. Communication of the ACM. Vol. 28. No. 1. 1985.
<POLY87> C.D. POLYCHRONOPOULOS,U. BANERJEE. Processor allocation for horizontal and vertical parallelism and related speedup bounds.IEEE Transactions on computers,vol.C-36,n°4,April 1987.
<OCCAM88> INMOS. OCCAM 2 Reference Manual. Prentice Hall, 1988.
<SOREL83> Sorel Y., Wolf P .Evaluation d'architectures de microprocesseurs de traitement du signal. Neuvième colloque sur le traitement du signal. Nice 1983.
<SOREL83> Sorel Y., Wolf P. Outils de mise au point de machines pour le traitement du signal. Etude d'un cas le modem 4800bps. Dixième colloque sur le traitement du signal. Nice 1985.
<TEXAS86> Texas Instrument. TMS32020 User's Guide. Digital Signal Processors Products.1986
<TRANS88> INMOS. Transputer Reference Manual. Prentice Hall, 1988.
<Wu84> Wu C.L.,Feng T.Y. Tutorial : Interconnection Network for Parallel and Multiprocessor System. IEE Computer Sciety Press, Silver Spring, 1984.