A Macrotask-level Unlimited Speculative Execution on Multiprocessors

Hayato YAMANA, Mitsuhisa SATO, Yuetsu KODAMA, Hirofumi SAKANE, †Shunichi SAKAI and Yoshinori YAMAGUCHI

Electrotechnical Laboratory, Computer Science Division
1-1-4 Umezono, Tsukuba, Ibaraki 305, Japan

†Real World Computing Partnership
1-6-1 Takezono, Tsukuba, Ibaraki 305, Japan

e-mail: yamana@etl.go.jp <URL: http://www.etl.go.jp/People/yamana/>
Tel: *+81-298-58-5955*, Fax: *+81-298-58-5882*

Abstract

The purpose of this paper is to propose a new fast execution scheme of FORTRAN programs. The proposed scheme enables the fast initiation of macrotask when its data dependences are satisfied even if the control flow has not been reached. The previous schemes to parallelize a program including conditional branches have a number of problems - **1)** Though the theoretical speedup ratio is up to N when N conditional branches are jumped on either a VLIW or a superscalar machine, the number of N is restricted up to the number of ALU's on a chip, **2)** Since conventional control schemes use a few processors to control macrotasks, the overhead to control them is large. The proposed scheme solves these problems - **1)** The proposed scheme enables speculative execution between coarse grain tasks, i.e., macrotasks, on multiprocessors by jumping many conditional branches, **2)** A distributed control scheme is proposed and implemented on the EM-4 multiprocessor to decrease the control overhead of macrotasks. Preliminary evaluations show that the control overhead of the proposed scheme is smaller than that of the other control schemes. Moreover, it is confirmed that the distributed control can be implemented by using software when the average macrotask execution time is larger than 14.4μs on the EM-4 multiprocessor.

Keywords: *Compiler, Distributed Control, Macrotask, Multiprocessor, Parallel Processing, Speculation, Speculative Execution*

1. Introduction

This paper proposes a macrotask-level unlimited speculative execution, which executes coarse grain tasks, called macrotasks, on multiprocessors with speculation. A macrotask is initiated when its data dependences are satisfied regardless of its control dependences' satisfaction. Previous works [RiFo72][NiFi84][LaWi92] reported that the speedup ratio is 12 to 630 times in comparison with conventional execution schemes without speculation when nonnumerical applications are executed.

Since conventional execution schemes keep both control dependence and data dependence, a statement cannot be initiated before the control flow is reached even if the input data are available. For example, S2 cannot be initiated before the conditional branch at S1 is executed to result in "taken."

```
If (A) then          S1
       B = C * D     S2
endif
```

However, if the data referenced at S2, C and D, are available before the execution of S1, S2 can be executed before the completion of S1 with speculation. After the execution of S1, S2 is validated according to the result at S1. The number of ignored conditional branches is called "speculation depth." When both the speculation depth and the computational resource are infinite, that is called ***oracle model***[NiFi84], the speedup ratio can attain up to 630 times[LaWi92].

Speculative execution schemes[LeSm84][AcKT86][SmLH90][ChLS92][SmLH92] have been studied mainly on a VLIW or a superscalar processor. Though the theoretical speedup ratio is up to N when the speculation depth is N, N is restricted up to the number of ALU's in the processor. That results in limited speedup with speculation. Therefore, speculation on multiprocessors, which have many ALU's, is required to attain high performance. Speculative execution schemes on multiprocessors[BaGa84][ThGH93], however, cannot extract full parallelism from a program.

The problems are both that **a)** a limited kind of loops is executed with speculation, and that **b)** the speculation path is restricted to result in limited speedup.

Banerjee and Gajski have proposed the speculative execution scheme [BaGa84] of Boolean recurrence loops[BaGa84] on the specialized multiprocessor, which consists of a special hardware called a Boolean recurrence solver. Though their paper is the first proposed paper of speculation on multiprocessors, only Boolean recurrence loops including only one conditional branch are executed with speculation.

Theobald, Gao and Hendren have proposed the speculative execution model[ThGH93] on multiprocessors, which performs speculation down only one path per branch to avoid increasing the number of program states. The model restricts the speculation depth to n. These restrictions, however, result in restricting the speedup ratio of speculation. As Urt has mentioned in [Uht92], the required computational resource will not explode when both the data dependence and control dependence are concerned. The main point of [Uht92] is that all the statements cannot be initiated even if control dependences are ignored, because some statements have data dependences that have not been satisfied. Besides that, Urt mentions that the control overhead to manipulate the initiation of statements exists. Therefore, the restrictions on speculation path proposed in [ThGH93] will be useless once the model is implemented on real multiprocessors.

This paper solves these problems by both **a')** generalizing the speculation and **b')** avoiding the restriction on speculation path.

In order to generalize the speculation on multiprocessors, we propose a macrotask-level speculative execution scheme. The macrotask, a set of statements, is a coarse grain task that is a unit of speculation. Although general parallelizing schemes among macrotasks have been proposed in [HoIK90][GiPo92], the advantages are **1)** the proposed scheme performs speculation, **2)** the macrotask is created so as to increase the effectiveness of the speculation, and **3)** the overhead to control macrotasks is reduced.

Moreover, the proposed scheme avoids the restriction on the speculation path. The speculation depth is dynamically determined depending on both the data dependences and the initiating overhead of a macrotask. When the initiating overhead is small, the speculation depth is eventually increased.

In the next section, the problems of speculative execution are reviewed and their solutions are described. Our execution models are described in Section 3. The creation scheme of macrotasks is explained in Section 4. A distributed control scheme is proposed and implemented on the EM-4 multiprocessor[SYHK89] in Section 5. The experimental results and their ramifications are presented in Section 6. The final section discusses the conclusions that may be drawn from this study, and describes our plans for future research.

2. Related Work and Our Approach

The basic purpose of speculation is to speedup program execution by running some code segments before it is known whether they are actually reached. The problems of speculative execution are:

a) Most schemes[LeSm84][AcKT86][SmLH90][ChLS92][SmLH92] have been studied on a VLIW or superscalar processor. Though the theoretical maximum speedup ratio is N when the speculation depth is N, the number of N is restricted up to the number of ALU's in a processor.

b) Speculative execution schemes on multiprocessors cannot be used with general programs, or they restrict the speedup ratio of speculation, since they handle a limited kind of loops[BaGa84], or restrict the speculation path[ThGH93].

We solve these problems by adopting the following approaches.

a') The speculation both inter and intra processors is performed to exploit full parallelism in a program.

b') We adopt a macrotask-level speculative execution to generalize the speculation. Moreover, the speculation path is not restricted so that the speculation depth is determined at runtime depending on both the data dependence and the initiating overhead of a macrotask.

Although general parallelization schemes among macrotasks have been proposed in [HoIK90][GiPo92], the schemes cannot be used directly with speculation since they adopt the centralized control that results in another problem:

c) The centralized control uses one or a few processors to control the macrotasks by updating the macrotask control table in which their execution conditions are stored. Therefore, the scheme increases the control overhead in proportion to the number of running macrotasks because the number of states in the macrotask control table increases in proportion to the number of running macrotasks.

The solution of the above problem is as follows.

c') A distributed control is proposed to decrease the macrotask control overhead. Each macrotask broadcasts and snoops the signals indicating determined branch directions, i.e., selected paths. The macrotask decides by itself whether to continue or to discard the execution when the macrotask receives the signal. Thus, the control overhead does not depend on the number of running macrotasks.

3. Background

In this section, the programming model and the platform are described. The definitions of basictask, macrotask, and control dependence are also described.

3.1 Programming Model

The program used in this paper must satisfy the following conditions:

1) The flow graph[AhSU86] of the program should be reducible[AhSU86], i.e., it has no jumps into a loop.

2) No subroutine calls exist, i.e., sub-programs should be expanded in line.

3) The conditional branches handling exceptions, such as divide by zero, are previously marked out to skip the following speculative execution because such speculation results in execution errors.

4) Data used in a program are stored in a shared memory or a distributed shared memory.

3.2 Platform

This paper assumes that a multiprocessor supports two mechanisms: a broadcasting mechanism and a dynamic task allocation mechanism. The broadcasting mechanism is used to broadcast the signals indicating the results of the conditional branches. The macrotask should be allocated dynamically to the processor.

3.3 Definition of Basictask and Macrotask

The basictask is defined as the code segment of a program that has only one entry point of control flow at the top of the segment. Thus, the smallest basictask consists of one statement. The macrotask is a set of basictasks that has only one entry point of control flow at its top. They may have plural exit points of control flow in them.

3.4 Definition of Control Dependence

Node y is control dependent on node x **iff** both node x and node y on the control flow graph satisfy the following two conditions.

c1) Node y does not post-dominate node x.

c2) There exist paths P from node x to node y. Node y post-dominates all the nodes in P except node x and node y.

The above definition shows that the node y is control dependent on node x, when the path from every node z to *exit* node includes node y while every path from node x to *exit* node does not include node y, where node z is the node on the path from node x to node y.

4. Creation of Macrotasks

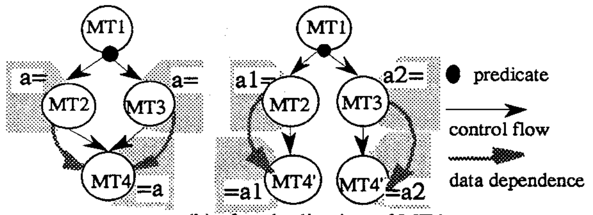
This section describes the algorithms to create macrotasks and shows the experimental results.

4.1 Macrotask for Speculation

Macrotasks are created both to **1)** prevent side-effects that result from double racing of the same variable, and to **2)** reduce the macrotask control overhead by enlarging their size.

1) Consider *MT2* and *MT3* in Fig.1(a), which define the same variable "*a*." If both *MT2* and *MT3* are initiated with speculation, the value of the variable "*a*" cannot be guaranteed to be safe, this can be called a "side-effect." To prevent side-effects from arising, the macrotask referencing the value "*a*" is duplicated as shown in Fig.1(b) so that data dependences between macrotasks are determined regardless of the conditional branch. After duplication, "variable renaming" is performed.

2) To enlarge the size of a macrotask, the basictasks are merged into one macrotask unless the effectiveness of speculation is lost after merging.



(a) original (b) after duplication of MT4

Fig.1 Duplicated macrotasks for preventing side effects.

4.2 Steps of Macrotask Creation

We can make macrotasks by adopting the following 4 steps.

1) Create basictask(BT) using HTG[GiPo92].

2) Create control flow graph, control dependence graph, and data dependence graph [GiPo92].

3) Duplicate basictasks to prevent side-effects.

4) Merge basictasks to create larger macrotask.

4.2.1 Creation of Basictask(BT) using HTG

A program is restructured into the hierarchical task graph(HTG) [GiPo92] to make the program to be a directed acyclic graph. The HTG is shown as HTG=(*HV*, *HE*) with unique nodes *START* and *STOP* belonging to *HV* such that there exists a path from *START* to every node in *HV* and a path from every node to *STOP*. Each node in *HV* is one of the following type:

a) simple node representing a task, i.e., basic block, that has no sub tasks,

b) compound node representing a task that consists of other tasks in an HTG, or

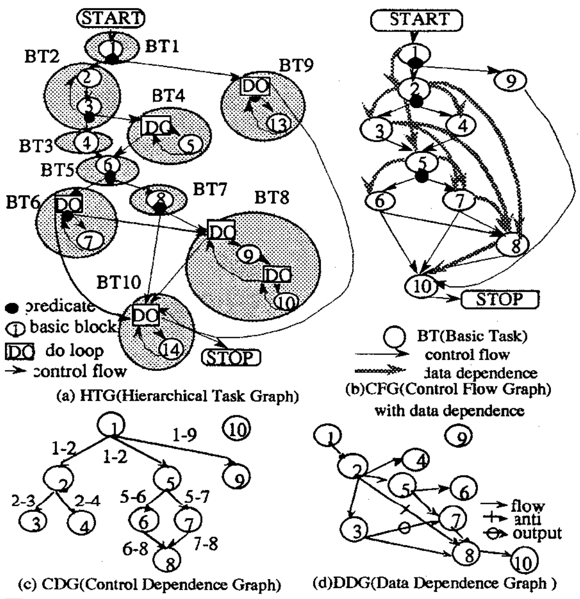
c) loop node representing a task that is a loop whose iteration body is an HTG.

An arc $he \in HE$ indicates control flow between nodes $hv \in HV$. An example of HTG is shown in Fig.2(a). The nodes, where at the same hierarchy level, form basictasks at that level. Fig.2(a) shows the basictasks (shown as BT) at the top hierarchy level.

4.2.2 Creation of CFG, CDG, and DDG[GiPo92]

A control flow graph is a directed graph CFG=(V, CFE) with unique nodes *START*, $STOP \in V$. A node $v \in V$ indicates the basictask. A directed arc $cfe(u, v) \in CFE$ indicates the control flow from *BTu* to *BTv*.

A control dependence graph is a directed graph CDG=(V, CDE) with labeled arcs. A node $v \in V$ shows a basictask. A directed arc $cde(u, v) \in CDE$ shows that *BTv* is control dependent on *BTu*. Thus, it is decided in *BTu* whether *BTv* is reached. A label *u-a* placed next to the arc $cde(u, v) \in CDE$ indicates that *BTu* decides the branch direction to *BTa* on the CFG. The label may have the value of *true* or *false*. For example, the label *u-a* has the value of *true* when the control flow reaches $cde(u, v)$. On the other side, the label has the value of *false* if the control flow does not reach $cde(u, v)$.



(a) HTG(Hierarchical Task Graph) (b)CFG(Control Flow Graph) with data dependence

(c) CDG(Control Dependence Graph) (d)DDG(Data Dependence Graph)

v	FDDS(v)	DN(v)	v	FDDS(v)	DN(v)
1	-	1,10	6	5	1,2,5,6,10
2	1	1,2,5,10	7	5	1,2,5,7,10
3	2	1,2,3,5,10	8	3,7	1,2,5,8,10
4	2	1,2,4,5,10	9	-	1,9,10
5	2	1,2,5,10	10	8	1,10

(e) FDDS(v) and DN(v) of (c)(d)

Fig.2 HTG, CFG, CDG, and DDG.

A data dependence graph is a directed graph DDG=(V, DDE), where an arc $dde(u, v) \in DDE$ shows the data dependence from BTu to BTv. There are three kinds of data dependence; flow dependence(read after write), anti-dependence(write after read), and output dependence(write after write). They are shown by using $fdde(u, v)$, $adde(u, v)$, and $odde(u, v)$, respectively.

These examples are shown in Fig.2(b)(c)(d).

4.2.3 Duplication of Basictasks to Prevent Side-effects

The basictasks are duplicated so that the data dependences are determined regardless of the direction of conditional branch. Since both the anti-dependence and the output dependence can be removed by performing "variable renaming," only the flow dependence is concerned. Both anti-dependence and output dependence are removed after the duplication.

The duplicating condition of node v is $\mathrm{FDDS}(v) \not\subseteq \mathrm{DN}(v)$, where $v \in V$ in DDG, $\mathrm{FDDS}(v)=\{u \mid fdde(u, v) \in DDE\}$, and the $\mathrm{DN}(v)$ is the set of nodes that are determined to be executed whenever node v is executed. $\mathrm{FDDS}(v) \not\subseteq \mathrm{DN}(v)$ indicates that the flow dependences to node v cannot be determined when node v is determined to be executed. $\mathrm{DN}(v)$ is led from $\mathrm{DN}(v)=\{x \mid x \text{ DOM } v, x \in V\} + \{y \mid y \text{ POSTDOM } v, y \in V\}$, where DOM and POSTDOM indicates "*dominate*" and "*post-dominate*", respectively. Node x *dominates* node v **iff** every path in CFG from *START* to node v contains node x[AhSU86]. A node always dominates itself. Node y *post-dominates* node v **iff** every path in CFG from node v to *STOP* (not including node x) contains node y [FeOW87].

The following is the definition of the calculation between labels and the steps to duplicate node v.

The calculation between labels uses one Boolean value and logical primitives, $\wedge$(and) and $\vee$(or). The label a-b has the value of *true* when the destination is decided to be node b at node a. Otherwise, the label a-b has the value of *false*. Every path $cde(a_i,a_{i+1})$ has the label a_i-c_i , which is *true* when the path $p(v)$ is selected, where path $p(v)=<a_0,a_1,..,a_n=v>, a_i \in V$. The set of labels belonging to the path $p(v)$ is shown by $L(p(v))=\{a_i\text{-}c_i \mid cde(a_i,a_{i+1}) \text{ has label } a_i\text{-}c_i, <a_i,a_{i+1}> \subseteq p(v)\}$. The set of the incoming paths to node v is shown by $P(v)=\{p(v) \mid p(v)=<a_0,a_1,..,a_n=v>, a_i \in V\}$. The definite control condition $CC(v)$, which guarantees the reachable control flow to node v, is defined as every label belonging to $L(p(v)), \exists p(v) \in P(v)$ has value of true. In the case of Fig.2, $CC(8)=(1\text{-}2 \wedge 5\text{-}6 \wedge 6\text{-}8) \vee (1\text{-}2 \wedge 5\text{-}7 \wedge 7\text{-}8)$. The negative condition of $CC(v)$ indicated by $\underline{CC(v)}$, which guarantees that the control flow does not reach node v, is defined as every label a-b belonging to $L(p(v)), \forall p(v) \in P(v)$ has the value of *false*. In the case of Fig.2, $\underline{CC(8)}=(\underline{1\text{-}2} \vee \underline{5\text{-}6} \vee \underline{6\text{-}8}) \wedge (\underline{1\text{-}2} \vee \underline{5\text{-}7} \vee \underline{7\text{-}8})$. When the number of branch destinations at node a is two, node b and node c, $\underline{a\text{-}b}$ is shown as $\underline{a\text{-}b}=CC(a) \wedge a\text{-}c$. By using the above relationship, $\underline{CC(8)}$ becomes $\underline{CC(8)}=1\text{-}9 \vee (1\text{-}2 \wedge 5\text{-}7 \wedge 7\text{-}10) \vee (1\text{-}2 \wedge 5\text{-}6 \wedge 6\text{-}10)$.

Tab.1 shows the definition of the symbols appearing in the above discussion. The duplication steps are shown in Tab.2. The latter explains the steps with an example of Fig.2.

Tab.1 Definition of Symbols

$dde(u,v)$	; the arc of data dependence from node u to node v on the DDG
$fdde(u,v)$	$=\{dde(u,v) \mid$ data dependence is flow dependence$\}$
$adde(u,v)$	$=\{dde(u,v) \mid$ data dependence is anti-dependence$\}$
$odde(u,v)$	$=\{dde(u,v) \mid$ data dependence is output dependence$\}$
$\mathrm{VN}(dde)$	; variable name of the dde
$\mathrm{cde}(u,v)$	; the arc of control dependence from node u to node v on the CDG
$p(v)$	$=<a0,a1,..,a_n=v>, a_i \in V$, where node $a0$ has no cde ; a path to node v on the CDG
$P(v)$	$=\{p(v) \mid p(v)=<a_0,a_1,..,a_n=v>, a_i \in V\}$; the set of paths to node v on the CDG
$L(p(v))$	$=\{a_i\text{-}c_i \mid \mathrm{cde}(a_i,a_i+1)$ has label $a_i\text{-}c_i, <a_i,a_{i+1}> \subseteq p(v)\}$; the set of labels belonging to $p(v)$
$\mathrm{DN}(v)$	$=\{x \mid x \text{ DOM } v, x \in V, v \in V\} + \{y \mid y \text{ POSTDOM } v, y \in V, v \in V\}$; the set of nodes that are executed whenever the node v is selected
$\mathrm{DDS}(v)$	$=\{u \mid dde(u,v)\}$; set of source nodes of $dde(u,v)$
$\mathrm{FDDS}(v)$	$=\{u \mid fdde(u,v)\}$; set of source nodes of $fdde(u,v)$
$CC(v)$	every label belonging to $L(p(v)), \exists p(v) \in P(v)$ is *true* ; the condition to guarantee the reaching of control dependence to node v
$\underline{CC(v)}$	$\exists a\text{-}b \in L(p(v))$ becomes false for $\forall p(v) \in P(v)$; the condition to guarantee the unreaching of control dependence to node v

Tab.2 Steps to duplicate basictasks

Step 1)	Calculate $\mathrm{FDDS}(v)$ and $\mathrm{DN}(v)$. For $\exists v$, $\mathrm{FDDS}(v) \subsetneq \mathrm{DN}(v)$, the following steps are applied. When no nodes satisfy $\mathrm{FDDS}(v) \subsetneq \mathrm{DN}(v)$, go to ***Step 8)***.
Step 2)	Calculate $\mathrm{NDS}(v)=\mathrm{FDDS}(v)-\{\mathrm{DN}(v) \cap \mathrm{FDDS}(v)\}$. Then, for $\exists u \in \mathrm{NDS}(v)$, apply the following steps.
Step 3)	Calculate $CC(u)$, $\underline{CC(u)}$, and $CC(v)$.
Step 4)	Discard node v from both CDG and DDG. Create two duplicated nodes $\langle v',v'' \rangle$ on both CDG and DDG.
Step 5)	Add the control dependence arc satisfying $CC(v')=CC(v) \wedge CC(u)$ to node v' on CDG. Add the control dependence arc satisfying $CC(v'')=CC(v) \wedge \underline{CC(u)}$ to node v'' on CDG. $CC(x)$ is indicated by using labels. $CC(x)=(a_{11}\text{-}b_{11} \wedge a_{12}\text{-}b_{12} \wedge .. \wedge a_{1(n1)}\text{-}b_{1(n1)}) \vee (a_{21}\text{-}b_{21} \wedge .. \wedge a_{2(n2)}\text{-}b_{2(n2)} \vee ... \vee (a_{m1}\text{-}b_{m1} \wedge .. \wedge a_{m(nm)}\text{-}b_{m(nm)})$, where and-or is reduced to be minimum Every "*or item*" shows a path to node x on CDG. Every "*and item*" shows a label of its path. Thus, add m paths to node x on CDG. Add the label $L(p_i(x))=\{a_{i1}\text{-}b_{i1}, a_{i2}\text{-}b_{i2},..,a_{i(ni)}\text{-}b_{i(ni)}\}$ to each path, where m is the number of "*or item*". Besides that, add the dummy node so that the control dependence arc outputs from node a when the label is a-b. The dummy nodes are used only for indicating the condition to satisfy the control dependence of node x
Step 6)	Add $dde(u,v')$ to node v' on DDG with the exception when $CC(u) \wedge CC(v')$ is always *true* for $\forall u \in \mathrm{DDS}(v)$. Add $dde(u,v'')$ to node v'' on DDG with the exception when $CC(u) \wedge CC(v'')$ is always *false* for $\forall u \in \mathrm{DDS}(v)$. Add $dde(v',w)$ and $dde(v'',w)$ for $\forall w \in \{w \mid dde(v,w) \in DDE\}$.
Step 7)	goto ***Step 1)***.
Step 8)	**Discarding non reachable fdde:** discard $\forall fdde \in \{fdde(u,v) \mid \exists w \in V, w \neq u, w \neq v, \mathrm{VN}(fdde(u,v))=\mathrm{VN}(fdde(w,v))=\mathrm{VN}(odde(u,w))\}$ from DDG.
Step 9)	**Discarding the duplicated nodes:** For $\exists v \in \{v \mid \exists u \in V, u \neq v, \mathrm{DDS}(v)=\mathrm{DDS}(u)$, both u and v have the same code which are duplicated fromthe same node$\}$, $\forall w \in \{w \mid dde(v,w)\}$, add $dde(u,w)$ on DDG and discard $dde(v,w)$. Add the control dependence arc satisfying $CC(u)=CC(v) \vee CC(u)$ on CDG. After that, discard node v from both DDG and CDG. Repeat the ***Step 9)*** until no v exist.
Step 10)	**Discarding the unnecessary dde.** Discard $\forall dde \in \{dde(u,v) \mid u \neq v, fdde(u,*)fdde(*,v)$ exits$\}$from DDG, where "$fdde(u,*)fdde(*,v)$" shows a flow dependence from node u to node v via at least one other node.
Step 11)	**Variable Renaming**· Rename $\mathrm{VN}(adde(w,v))$ for $\forall v \in \{v \mid \exists w \in V, w \neq v, adde(w,v)\}$. Rename $\mathrm{VN}(odde(w,v))$ for $\forall v \in \{v \mid \exists w \in V, w \neq v, odde(w,v)\}$. Then, all of the $adde$ and $odde$ are discarded

Firstly, ***Step 1)*** is applied to lead the duplication target nodes {8,10} shown in Fig.2(e). Then, ***Step 2)*** is applied to one of the nodes, node 8, to lead NDS(v). Then, we have NDS(8)={3,7} that shows the data dependences from both node 3 and node 7 to node 8 are uncertain. ***Step 3)*** is applied to the nodes belonging to NDS(8). Firstly, ***Step 3)*** is applied to node 3. Then, we have CC(3)=1-2∧2-3, $\underline{\text{CC(3)}}$=1-9∨(1-2∧2-4), CC(8)=(1-2∧5-6∧6-8)∨(1-2∧5-7∧7-8). After applying ***Step 4)*** and ***Step 5)***, we have CC(8')=CC(8)∧CC(3)=(1-2∧2-3∧5-6∧6-8)∨(1-2∧2-3∧5-7∧7-8), CC(8")=CC(8)∧$\underline{\text{CC(3)}}$=(1-2∧2-4∧5-6∧6-8)∨(1-2∧2-4∧5-7∧7-8), where nodes 8' and 8" are the duplicated nodes. After the duplication, control dependence arcs for nodes 8' and 8" are added. ***Step 6)*** adds the data dependence arcs to them shown in Fig.3. The ***Steps 1)*** to ***7)*** are applied until no nodes v satisfying FDDS(v)$\not\subset$DN(v) exist. Finally, we have the CDG and DDG shown Fig.4.

Next, ***Step 8)*** is applied to discard the useless data dependences. For example, *fdde*(3,8') is discarded when *fdde*(3,8'), *fdde*(7,8'), and *odde*(3,7) show the same variable, because the variable is redefined at node 7. Since DDS(8') and DDS(8") become identical after ***Step 8)***, node 8' and node 8" are merged into one node shown as node 8' after applying ***Step 9)***. In the same way, node 10' and node 10" are merged into node 10'. After applying ***Step 10)***, *adde*(2,8') and *adde*(2,8"") are discarded. ***Step 11)*** renames the variables to discard *odde*(3,7) and *adde*(2,8"'). Moreover, the variables in the duplicated nodes (8',8"',8"",10',10"',10"" ,10""') are renamed to avoid side-effects. Finally, we have CDG and DDG shown in Fig.5.

4.2.4 Creation of Macrotask

Basictasks may be merged into one macrotask to enlarge the size, when the following basictask has both data dependence and control dependence from the preceding basictask, because the following basictask cannot be initiated before the completion of the preceding basictask. The necessary and sufficient condition that keeps speculation effective is unchanging of the satisfaction time of data dependences, because the initiation time of macrotask is delayed if the satisfaction time of data dependences is delayed. Besides that, the merged macrotask must have only one entry point of control flow, which is defined in 3.3.

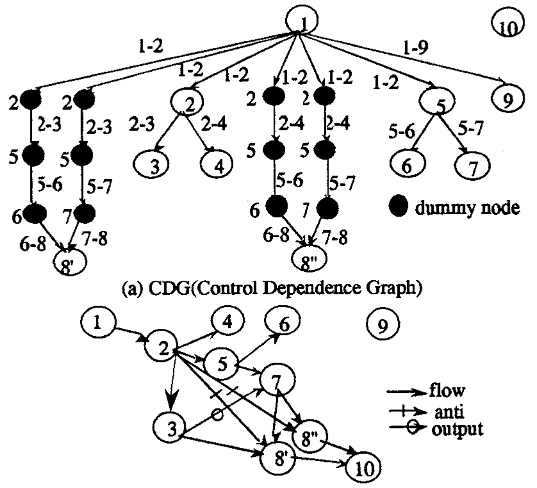
Firstly, the data dependences both to *BTa* and to *BTb* must satisfy at least one of two conditions to guarantee the unchanging of the satisfaction time of data dependences:

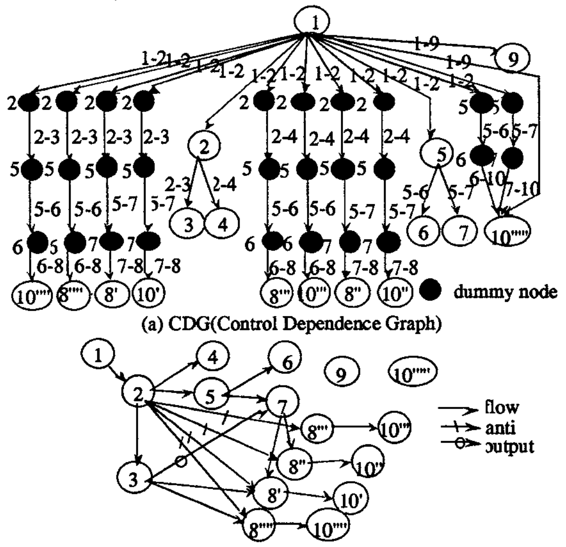
$$\mathrm{DDS}(BTa)=\mathrm{DDS}(BTb) \qquad (1)$$

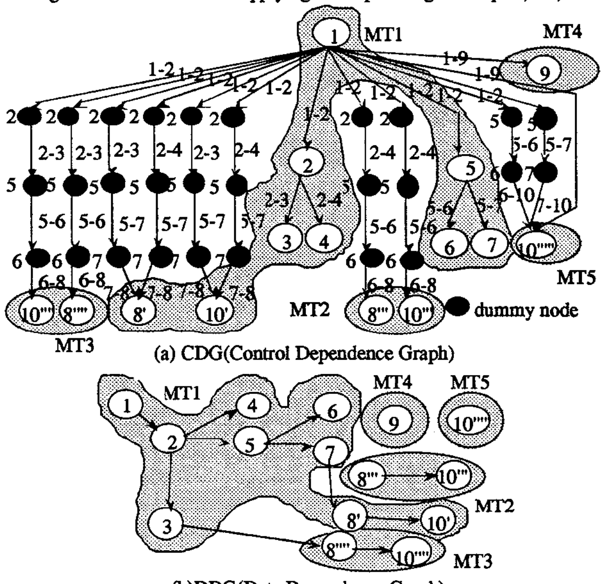
$$\exists dde(BTa,BTb), \forall BTx, BTx \in \{BT \mid BT \neq BTa, BT \in \mathrm{DDS}(BTb)\} \Rightarrow dde(BTx,*)dde(*,BTa) \qquad (2)$$

where *dde*(*BTx*,*)*dde*(*,*BTa*) shows that the data dependence from *BTx* to *BTa* exists through any *BT*.

Equation (1) guarantees the data dependences both to *BTa* and *BTb* are identical. Equation(2) guarantees that the data dependences to *BTb* are satisfied when the *BTa* completes its execution. This means the satisfaction time of the merged two basictasks becomes *BTa*'s satisfaction time.


dummy node
(a) CDG(Control Dependence Graph)
flow
anti
output
(b)DDG(Data Dependence Graph)
Fig.3 CDG & DDG after duplicating node 8


dummy node
(a) CDG(Control Dependence Graph)
flow
anti
output
(b)DDG(Data Dependence Graph)
Fig.4 CDG & DDG after applying the duplicating BT *steps 1) - 7).*


MT1
MT2
MT3
MT4
MT5
dummy node
(a) CDG(Control Dependence Graph)
(b)DDG(Data Dependence Graph)
Fig.5 CDG & DDG after applying the duplicating BT *step 11)* and the created macrotasks.

Secondly, the control dependences both to *BTa* and to *BTb* must satisfy at least one of two conditions to guarantee the merged basictasks to be a macrotask:

$$\mathrm{CC}(BTa)=\mathrm{CC}(BTb) \quad (3)$$

$$\mathrm{CC}(BTa)\wedge\{a\text{-}x\}=\mathrm{CC}(BTb) \quad (4)$$

where $\{a\text{-}x\}$ shows a partial set of the labels determined whether *true* or *false* in *BTa*.

Equation (3) guarantees the control dependences both to BTa and to BTb are identical. Equation (4) guarantees that the control dependences to *BTb* are guaranteed to be satisfied by both the control dependences of *BTa* and the control dependences satisfied in *BTa*.

Therefore, the conditions to merge two basictasks are that both one of (1)(2) and one of (3)(4) are satisfied. We have four combinations of the conditions. An example is shown in Fig.5 that shows the 5 merged macrotasks out of 15 basictasks.

4.3 Experimental Results of Macrotask Creation

The programs used in this experimentation are shown in Tab.3. Four scientific calculation sub-programs are used. Firstly, these programs are restructured into HTG's. Then, basictasks are converted to macrotasks at each hierarchy level. Three of four programs have two hierarchy levels and the other has three hierarchy levels.

The number of basictasks and macrotasks are shown in Tab.3. Since the total program size increases after the macrotask creation because some basictasks are copied to prevent side-effects, the sizes of programs before and after the macrotask creation are also shown. This experimentation shows that the size of macrotasks is about three times larger than that of basictasks at the bottom hierarchy level. At the next hierarchy level, we have larger macrotasks. Thus, it is confirmed that the proposed scheme able to increase the size of macrotasks to decrease the macrotask control overhead.

Table 3 A Comparison of the size of Basicktask and Macrotask

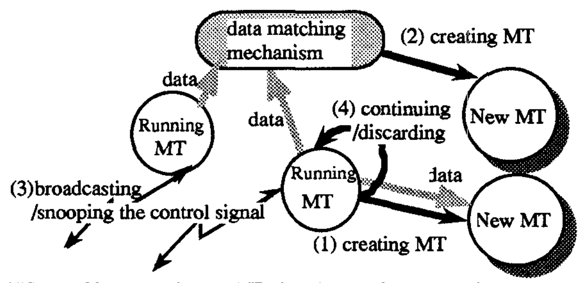
Program			Minimizing f(x) using Quadratic Interpolation	Muller's method	Orthogonal transformation of eigenvectors	Gaussian elimination method with partial pivoting
original	# of total lines		399	179	337	235
	# of BT		133	63	143	74
	# of lines / BT		3.0	2.8	2.4	3.2
after creating MT	# of total lines		589	269	612	280
	bottom level	# of MT	71	32	63	26
		# of lines / MT	8.3	8.4	9.7	10.8
	2nd level	# of MT	17	3	33	13
		# of lines / MT	34.6	90.0	18.5	21.6
	3rd level	# of MT				4
		# of lines / MT				70.0

5. Distributed Control Scheme of Macrotasks

In this section, a new control scheme of macrotasks is proposed to reduce the control overhead. The scheme is used at each hierarchy level described in 4.2.1.

5.1 The Distributed Control Scheme

The proposed distributed control scheme initiates the macrotask when its input data dependences are satisfied. The outline is shown in Fig.6. Each running macrotask, which has its own assumed path, snoops the broadcasted control signals indicating the selected paths on conditional branches. When the macrotask decides the assumed path is not selected, it discards the execution, while it continues the execution when the assumed path is selected. Therefore, each macrotask can determine the next state by itself, that makes no overhead depending on the number of macrotasks.



*(1)*Source *MT* creates the *new MT* when the *new MT* has one data dependence from the *source MT*. *(2)*The data matching mechanism creates the *new MT* when the all the data dependences are satisfied. *(3)*Every *MT* broadcasts the branch direction. *(4)* Every *MT* controls itself to select or to discard depending on the broadcasted control signals.

Fig.6 A distributed control scheme.

The following four mechanisms are indispensable to implement the proposed scheme:

1) Data matching mechanism is used to guarantee all the data are defined when a newly initiated macrotask has data dependences from many other macrotasks.

2) Broadcast mechanism is used to broadcast the signals indicating the completion of the macrotask and the selected paths.

3) Dynamic task allocation is used to allocate the newly initiated macrotask to the low loaded processor.

4) Processor control mechanism is used to determine the next state of the running macrotasks by snooping the control signals.

We have implemented the proposed scheme on the EM-4 multiprocessor[SYHK89]. In the implementation, we use the mechanisms of the EM-4 to decrease the control overhead; (1) direct matching mechanism[YaSK91] is used for the data matching, (2) circular omega network [YaSK91] is used as the broadcast mechanism, and (3) a MLPE packet, detecting the load-minimum PE [YaSK91], is used for dynamic load balancing. As for the processor control mechanism, we have implemented it by software.

5.2 Initiation of Macrotask Execution

The initiating condition of a macrotask is that all the data referenced in the macrotask are defined. The following are the details.

1) The macrotask with no data dependences from other macrotasks is able to be initiated at any time. A dummy macrotask is created to initiate such macrotasks.

2) The macrotask with data dependences only from one

macrotask is able to be initiated by its source macrotask which has the initiation code set.

3) The macrotask with data dependences from many macrotasks is able to be initiated after guaranteeing all the data dependences are satisfied. The data matching mechanism is used to wait for all the data definitions and to initiate the macrotask.

5.3 Control Scheme of Macrotask

The macrotask is controlled by using three parameters: level number L, assumed control path P, and maximum speculation depth N. Every running macrotask, having P and L, makes a decision whether to continue or to discard the execution, after comparing the assumed control path P and the broadcasted control signal.

The level number L shows the number of determined branch directions, that is set to zero at the beginning of the execution. The assumed control path P shows the path from the root macrotask whose control flow is determined. The assumed path P consists of the branch direction T, F, and *, that represents "*true*", "*false*," and "*don't care*" respectively. The root macrotask has null assumed path. The other macrotasks have plural assumed paths from the root macrotask. The maximum speculation depth N, which comes from the hardware resource limitation, is used to limit the speculation depth.

In the example shown in Fig.7, *MTi* is the root macrotask. *MTk*, which has P=*FT* and L=*L*, is on speculation, where the conditional branch in *MTi* is the *L*th branch. When *MTn* is data dependent only on *MTk*, *MTk* is able to initiate *MTn*, which has the same level number L=*L* and the assumed path P=(*MTk's* assumed path)+(Path from *MTk* to *MTn*). The level number L remains the same when the root macrotask does not change.

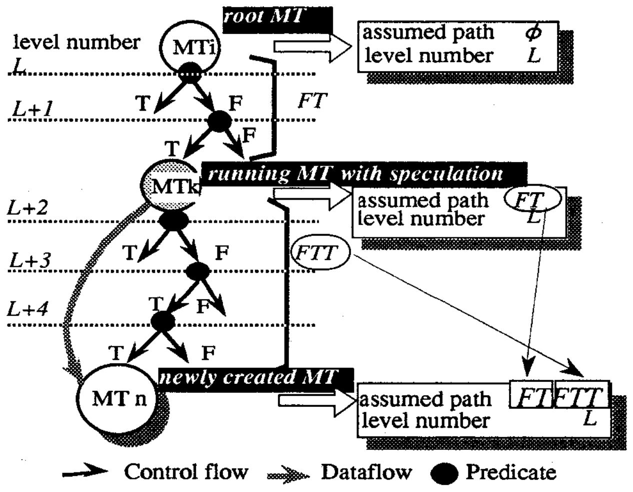
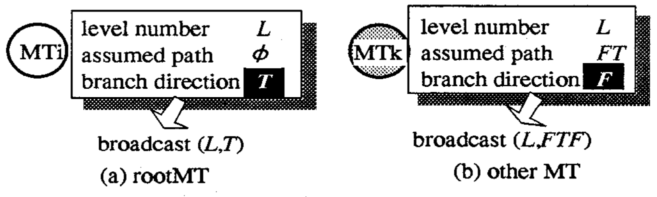


Fig.7 Assumed path and level number.



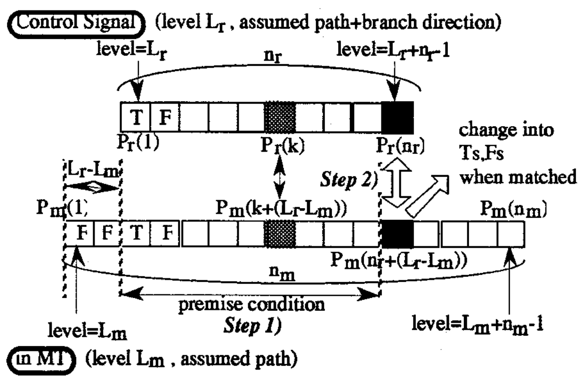
control signal=(*level number*, *assumed path+branch direction*)

Fig.8 Broadcasted control signal

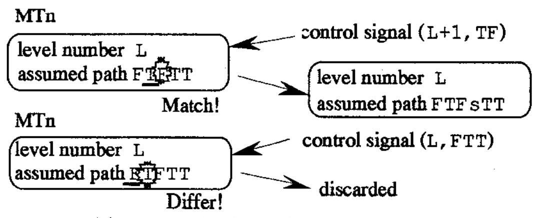
The execution sequence is described as follows. As shown in Fig.8, the control signal consisting of (level number L, assumed path P + branch direction) is broadcasted when the branch condition is determined. For example, the control signal (*L,T*) is broadcasted when the branch condition is determined as *true* at the root macrotask whose level is *L*. On the other side, the signal (*L,FTF*) is broadcasted when the branch condition is determined as *false* at the speculated macrotask whose level is *L* and assumed path P=*FT*. The control signal (*L,FTF*) shows that the far left side of *FTF* is the *L*th branch and the false is determined on the assumption that the path P=*FT* will be selected.

The received control signal is shown as follows: the level number is L_r, the number of Boolean in the (assumed path P + branch direction) is n_r, and the kth Boolean form the far left side of (assumed path P + branch direction) is $P_r(k)$, where $1 \le k \le n_r$. In the same way, the strings L_m, n_m, $P_m(k)$ are defined for the same value in the macrotask running on the processor.

When the assumed paths of the macrotask running on the processor have the path corresponding to the branch direction $P_r(n_r)$ of the received control signal, the processor executes the following 5 steps. Otherwise, the received control signal is discarded. That is, when $L_m < L_r + n_r \le L_m + n_m$ is satisfied, the processor executes the following ***Step 1) to 5)***. $L_r + n_r - 1$ shows the level number of $P_r(n_r)$. $L_m + n_m - 1$ shows the level number of $P_m(n_m)$. When $L_r + n_r - 1 < L_m$ is satisfied, the received control signal is useless because $P_r(n_r)$ corresponds the definite path (not an assumed path) of the running macrotask. When $L_m + n_m - 1 < L_r + n_r - 1$ is satisfied, the received control signal is useless because $P_r(n_r)$ corresponds the future path beyond the assumed path of the running macrotask.



(a) comparison between the control signal and assumed path in MT



(b) an example of updating the assumed path

Fig.9 An Updating of the assumed path.

Every macrotask keeps the past definite path up to the maximum length N-1. The past definite path is used to compare with the assumed path of the received control signal when $L_r<L_m$. Since $L_m<L_r+n_r$ and the maximum number of n_r is N, we have L_m-L_r<N. Thus, the length N-1 is enough.

***Step 1)* Comparison of Premise Condition**

As shown in Fig.9, $P_r(k)$ corresponds to $P_m(k+(L_r-L_m))$. $P_r(k)$ and $P_m(k+(L_r-L_m))$ are compared in the scope of $1 \le k \le n_r-1$, where they are same in the case of (T,T),(F,F),$(*,F)$,$(*,T)$, $(*,*)$,$(F,*)$ or $(T,*)$. Only in case of $(F,*)$ and $(T,*)$, the assumed path of the macrotask is duplicated and following steps are applied to the duplicated path because $P_m(k+(L_r-L_m))$ is not fixed though $P_r(k)$ is fixed as T or F. When $\forall k, 1 \le k \le n_r-1$, $P_r(k)$ is the same as $P_m(k+(L_r-L_m))$, the latter steps are applied because it is confirmed that the received control signal and the assumed path are on the same path. Otherwise, the received control signal is discarded.

***Step 2)* Comparison of Branch Direction**

$P_r(n_r)$ is compared with the corresponding assumed path $P_m(n_r+(L_r-L_m))$. If they are same, $P_m(n_r+(L_r-L_m))$ is checked as Ts or Fs, that shows the "*selection*" of the corresponding assumed path $P_m(n_r+(L_r-L_m))$. The macrotask, however, discards speculation when they are different. An example is shown in Fig.9(b).

***Step 3)* Updating the Level**

The assumed paths from $P_m(1)$ to $P_m(j)$ are discarded when every $P_m(k)$, $1 \le k \le j$ is one of Ts and Fs, where $\exists j$, $1 \le j \le n_m$. After discarding, the level number is incremented by j, because the branch directions to jth level have been confirmed. When the n_m becomes *zero*, the macrotask is now a root macrotask.

***Step 4)* Initiating Subsequent Macrotasks**

A new macrotask is initiated by its source macrotask, a dummy macrotask, or a data matching mechanism described in 5.2, when the length of the assumed path, e.g. the length of *FTFTT* is 5, is less than the maximum speculation depth N. Since the subsequent macrotask whose assumed paths have been invalidated by the previous control signals is not initiated, the speculation depth is determined dynamically depending on both the macrotask control overhead and data dependences. That means the speculation depth increases when the macrotask control overhead is small and there exist small number of data dependences.

***step 5)* Reclaim of Level Numbers**

Level numbers must be reclaimed to avoid the exhaustion of level numbers because the maximum level number is limited by hardware resource. The maximum level number is shown as L_{max}. The level numbers are divided into two groups: 0 to L_{mid} and L_{mid}+1 to L_{max}, where L_{mid} is the middle number of L_{max}. The flag included in the control signal, indicating "the reclaim of the level numbers to L_{mid}," is checked when the root macrotask broadcasts the control signal with the level number L_{mid}+1. Every processor receiving the control signal with the checked flag sends an acknowledge signal to the predefined processor after confirming that all the level numbers of running macrotasks in its processor exceed the number L_{mid}+1. Then, the predefined processor broadcasts the signal, indicating that the level numbers 0 to L_{mid} are available for use, after receiving all the acknowledgements. After that, every processor is again able to use the level numbers from 0 to L_{mid}. In this way, the level numbers are reclaimed and reused. Since the level numbers are reclaimed half by half, the macrotasks are executed without interruption.

5.4 Comparison with Centralized Control

The control signal is sent as "the processor executing a macrotask ⇒ the control processor ⇒ the target processor to control" when the centralized control is adopted. The control signal of the distributed control, however, is sent as "the processor executing a macrotask ⇒ all the processors." When a new macrotask is initiated, the control signal is sent as "the macrotask including the initiation code set to create a new macrotask ⇒ the target processor to assign the macrotask." The macrotask control overhead is defined as the time from sending a control signal at a macrotask until controlling the other macrotasks. The macrotask control overhead is shown in Tab.4, where C_c and C_d show the centralized control overhead and distributed control overhead respectively. T_1 shows the communication delay of one to one communication. T_b shows the communication delay of broadcasting. The value of C_c increases in proportion to the number of running macrotasks as described in 2.. The value of C_d, however, is constant regardless of the number of running macrotasks.

Table 4 A comparison of the centralized control and the proposed distributed control.

	Centralized Control	Distributed Control
Macrotask Control Overhead	$2T_1+C_c$	C_d+T_1(in case of creating MT) T_b+C_d(in case of discarding MT and updating the level)

T_1: one to one comunication delay (3.3μs in EM-4)
T_b: broadcasting comunication delay (26.4μs in EM-4)
C_c: centralized control overhead ($\propto$ the number of MT)
C_d: distributed control overhead (constant)

6. Evaluations

The distributed control has been implemented on the EM-4 multiprocessor[SYHK89] to evaluate the decrease of the macrotask control overhead.

6.1 EM-4 multiprocessor and implementation

The EM-4 multiprocessor[SYHK89] consists of 80 processing elements whose machine clock cycle is 12.5MHz. The single chip processor called EMC-R has the fast synchronous communication mechanism between threads. The instruction is completed in one cycle except load, store, multiplier, and divide. The interconnection network adopts circular omega topology whose performance is 60.9Mbytes/sec per link.

The macrotask control mechanism is written in software using EM-C, a parallel extension of C language with thread communication libraries. The centralized control uses three processors to control the remaining 77 processors. The distributed control, however, is implemented on all the processors that receive and send the control signals. Fig.10 shows the outline of the implementation of the distributed control onto the EM-4. The distributed control is programmed by 7 threads; three of them are used for controlling macrotasks, the rest four of them are the threads of macrotask themselves. These 7 threads are allocated to every processor. In this implementation, the maximum speculation depth N is 32, the maximum number of level L_{max} is 2^{23}-1. The value of T_l is 3.3μs and T_b is 26.4μs, which is the processor to processor communication time and the broadcast time, respectively.

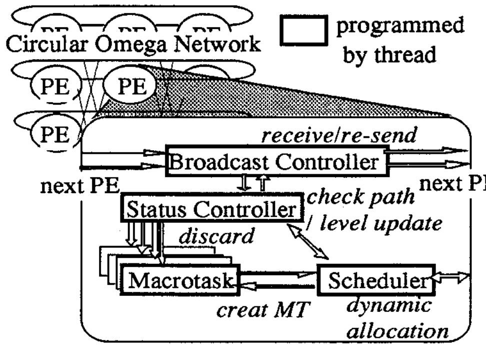


Fig.10 An implementation of the distributed control on the EM-4

6.2 Workload

The evaluated program is shown in Fig.11. The program includes a Boolean recurrence loop. The data dependence remains continuously when the same direction is selected at each iteration. However, it breaks once the another direction is selected. The macrotasks are created by using the proposed macrotask creation scheme.

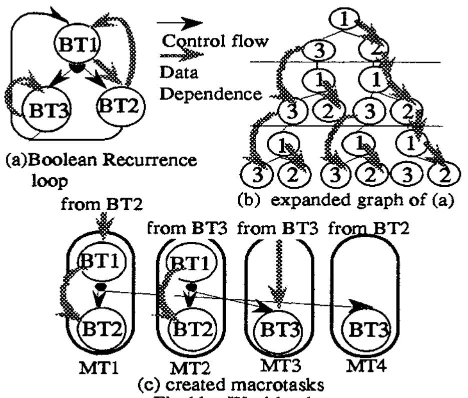


Fig.11 Workload.

6.3 Speculation with Various Task Size

We assume that **1)** every BT has the same task size, 0.96μs×*TaskSize*, in which the dummy calculation is performed, **2)** a 32bit data is transformed between BT's when they have data dependence, **3)** the number of iteration is 10, and **4)** the conditional branches are selected in the sequence of *BT1,BT2,BT1,BT3,BT1,BT2,...* The theoretical speedup ratio with no overhead on this sequence is 6.67.

The speedup ratio is shown in Fig.12. The speedup curve of the proposed distributed control scheme is sharper than that of the centralized control scheme. This observation shows there is a small overhead to control macrotasks. The speedup ratios of both execution schemes, however, are less than 1.0 when TaskSize is less than 15. That means the control overhead is larger than the time shortened by the speculation when TaskSize is less than 15, i.e., 14.4μs.

Since the average execution time of macrotasks at the bottom hierarchy level in Tab.3 is between 7μs and 9μs, it is still small to hide the control overhead. However, the average execution time of macrotasks at second hierarchy level is between 150μs and 300μs when the programs are executed with the matrix whose size is 10, because the macrotasks at second hierarchy level include loops. Therefore, the software implementation of the distributed control is able to be adopted when the macrotasks at second or higher hierarchy levels are controlled. The macrotasks at the bottom hierarchy level, however, should be controlled by hardware to hide the control overhead when the average execution time of macrotasks is less than 14.4μs.

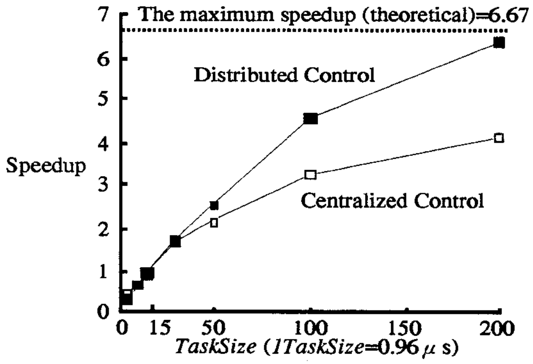


Fig.12 Speedup versus *TaskSize*.

6.4 Speculation with Various Numbers of Macrotasks

The number of iteration is varied to change the number of running macrotasks. We assume that **1)** TaskSize is fixed at 50, **2)** a 32bit data is transformed between BT's when they have data dependence, **3)** the number of iteration is varied from 5 to 30 times, and **4)** the selected path is same as that in **6.3**. The number of running macrotasks is calculated by 4 macrotasks × the number of iteration.

Since both *MT2* and *MT4* at each iteration have no input data dependences, these macrotasks are initiated with speculation at the beginning. Although the theoretical speedup ratio with no overhead is (the number of iteration) × 0.67, the macrotask creation overhead at the beginning bounds the maximum speculation depth to 10, i.e., the maximum speedup ratio is 6.67. That means *MT2* and *MT4* at iteration 11 will be created after the determination of the conditional branch at iteration 1. The result is shown in Fig.13.

The speedup ratio increases when the iteration number is less than 10 with both the schemes, because the speculation depth increases in proportion to the number of iterations. The situation changes after exceeding 10 iteration times. The speedup ratio decreases with the centralized control, while the ratio holds the same level with the distributed control. This observation shows that the macrotask control overhead of the centralized control increases in proportion to the number of running macrotasks as shown in Tab.4. Therefore, the speedup ratio of the centralized control decreases when the number of iteration increases, after the iteration 10. However, the speedup ratio of the distributed control holds the same level because the macrotask control overhead is independent on the number of running macrotasks.

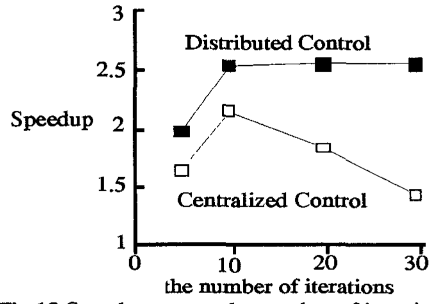


Fig.13 Speedup versus the number of iterations(*TaskSize*=50).

7. Conclusions

In this paper, we propose the unlimited speculative execution that enables the speculation inter processors to exploit rich parallelism. The scheme uses two new mechanisms : the macrotask creation and the distributed control.

The advantages of the macrotask creation are 1) preventing side-effects, and 2) decreasing the control overhead of macrotasks by enlarging the size of macrotask. The preliminary experiments show that the scheme is able to enlarge the size about 3 times larger than that of basictask.

The distributed control achieves small overhead to control macrotasks. The scheme has been implemented on the EM-4 multiprocessor[SYHK89] by using software. The preliminary evaluation shows that the macrotask control overhead is smaller than that of other macrotask control schemes. Moreover, it is confirmed that the software implementation is able to be adopted when the macrotasks at second or higher hierarchy levels are controlled. However, the evaluation shows that the hardware to control macrotasks is necessary to control macrotasks at the bottom hierarchy level of HTG when the average execution time of macrotasks is less than 14.4μs, because the software implementation still has large overhead. Thus, we are now estimating the control overhead when the control hardware is available. Furthermore, we are considering a macrotask scheduling algorithm to use point-to-point signaling, which decreases the control overhead, instead of broadcasting.

Our goal is to perform speculation on multiprocessors by using the distributed control, and to produce an automatic compiler, regenerating an original program to the program divided into macrotasks. We are now constructing its HPF compiler.

Acknowledgement

The authors would like to thank Dr. Kimihiro OHTA, Director of the Computer Science Division for supporting our research.

References

[AcKT86] R.D.Acosta, J.Kjelstrup and H.C.Torng:"An Instruction Issuing Approach to Enhancing Performance in Multiple Functional Unit Processors," IEEE Trans. Comput.,35, 9,pp.815-828 (1986).

[AhSU86] A.V.Aho, R.Sethi and J.D.Ullman: "Compilers: Principles, Techniques and Tools," Addision-Wesley (1986).

[BaGa84] U.Banerjee and D.D.Gajski : "Fast Execution of Loops with IF Statements," IEEE Trans. Comput., **C-33**, 11, pp.1030-1033 (1984).

[ChLS92] M.C.Chang, F.Lai and R.J.Shang: "Exploiting Instruction-Level Parallelism with the Conjugate Register File Scheme," Proc. of 25th Ann. Int. Symp. on MicroArchitecture, pp.29-32 (1992).

[FeOW87] J.Ferrante, K.J.Ottenstein, and J.D.Warren:"The Program Dependence Graph and Its USE in Optimization," ACM Trans.Prog.Lang.Syst.,**9**, 3, pp.319-349 (1987).

[GiPo92] Milinf Girkar and C.D.Polychronopolulos: "Automatic Extraction of Functional Parallelism from Ordinary Programs", IEEE Trans. Parallel & Distributed Syst.,**3**, 2, pp.166-178 (1992).

[HAOK92] H.Honda, K.Aida, M.Okamoto and H.Kasahara :"Coarse Grain Parallel Execution Scheme of a Fortran Program on OSCAR", IEICE Trans., **J75-D-I**, 8, pp.526-535 (1992)*(in Japanese).*

[HoIK90] H.Honda, I.Iwata and H.Kasahara:"Coarse Grain Parallelism Detection Scheme of a Fortran Program", IEICE Trans.,**J73-D-I**, 12, pp.951-960 (1990)*(in Japanese).*

[LaWi92] M.S.Lam and R.P.Wilson: "Limits of Control Flow Parallelism," Proc. of Ann. Symp. on Comput. Architecture, pp.46-57 (1992).

[LeSm84] J.Lee and A.J.Smith:"Branch Prediction Strategies and Branch Target Buffer," IEEE Comut.,**17**,1,pp.6-22 (1984).

[NiFi84] A.Nicolau and J.A.Fisher:" Measuring the Parallelism available for Very Long Instruction Word Architecture", IEEE Trans. Comput., **33**, 11, pp.968-976 (1984).

[RiFo72] E.M.Riseman and C.Foster: "The Inhibition of Potential Parallelism by Conditional Jumps," IEEE Trans. Comput.,**21**,12, pp.1405-1411 (1972).

[SmLH90] M.D.Smith, M.S.Lam and M.A.Horowitz: "Boosting Beyond Static Scheduling in a Superscalar Processor," Proc. of Ann Symp. on Comput. Arch., pp.344-344 (1990).

[SmLH92] M.D.Smith, M.S.Lam and M.A.Horowitz: "Efficient Superscalar Performance Through Boosting," Proc. Int. of Conf on ASPLOS-V, pp.248-259 (1992).

[SYHK89] S.Sakai,Y.Yamaguchi,K.Hiraki, Y.Kodama,T.Yuba :"An Architecture of a Dataflow Single Chip Processor", Proc. of 16th Ann. Symp. on Computer Architecture, pp.46-53(1989).

[ThGH93] K.B.Theobald, G.R.Gao, and L.J.Hendern: "Speculative Execution and Branch Prediction on Parallel Machines," Proc. Int. Conf. of Supercomputing, pp.77-86 (1993).

[Uht92] A.K.Uht: "Requirements for Optimal Execution of Loops with Tests," IEEE Trans. Parallel and Distrib. Syst., **3**, 5 pp.573-581 (1992).

[YaSK91] Y.Yamaguchi,S.Sakai and Y.Kodama :"Synchronization Mechanisms of a highly parallel dataflow machine EM-4", IEICE Trans., **E74**, 1, pp.204-213(1991).