様　式　Ｃ－１９、Ｆ－１９、Ｚ－１９　（共通）

科学研究費助成事業　　研究成果報告書

平成　２７　年　　６　月　　９　日現在

機関番号：　１４４０１
研究種目：　基盤研究(C)
研究期間：　2012～2014
課題番号：　２４５６０４５８

研究課題名（和文）ネットワーク符号化における線形誤り訂正符号の結合重み分布を用いた誤り特性評価

研究課題名（英文）Performance Analysis of error correcting code in network coding using joint weight distribution

研究代表者

藤原　融（Fujiwara, Toru）

大阪大学・情報科学研究科・教授

研究者番号：７０１９００９８

交付決定額（研究期間全体）：（直接経費）　　4,200,000円

研究成果の概要（和文）：情報を効率よく伝送する手法としてネットワーク符号化の理論が知られている。ネットワーク符号化の高信頼化のためには、効率化のために符号化されたデータを誤り訂正符号により再度符号化して、通信で生じる誤りを訂正する。本研究では、研究代表者によるネットワーク符号化向け誤り訂正符号に関する従来研究をもとに、結合重み分布の計算法の確立を行い、それを用いた誤り特性評価を行う方法を検討した。本報告では主として結合重み分布の計算について、GPUを用いた高速化や符号の構造を用いた高速化について述べる。

研究成果の概要（英文）：Network coding is studied for efficient communication in networks. For reliable communication, error correcting code is used to encode data of network coding. In this research, we have developed the computation method for joint weight distribution, and established evaluation methods for the performance analysis of error correcting codes. In this report, we mainly describe the computation of joint weight distribution using GPU and code structure.

研究分野：　情報理論、符号理論、情報セキュリティ

キーワード：　ネットワーク符号化　誤り訂正符号　符号化　重み分布　結合重み分布

１．研究開始当初の背景

誤り訂正符号の復号法や性能解析については、さまざまな研究が行われている。性能解析の基本となるのは符号の重み分布であるが、分割重み分布など、より詳細な重み構造が得られれば、より正確な性能解析を行うことができる。申請者もこれまで長年にわたり、符号の構造（重み構造やトレリス構造）に関する研究を行っている。

一方、ネットワーク符号化の研究も Ahlswede らによる 2000 年の研究以来、近年盛んに行われている。ネットワーク符号化技術は、中継ノードでデータの演算（複数のデータベクトルの線形和をとる等）を許すことにより、通信効率を上げようとするものである。以下では、まず、ネットワーク符号化について簡単な例を用いて説明する。

図 1 のようなネットワークにおいて、ソースノード s から２つのシンクノード t_1, t_2 への伝送を考える。各エッジは単位時間当たり１ビット伝送できるとする。ビット a と b の２ビット伝送するとき、図のように中間ノード３が受信データ a, b からその線形和 $a+b$ を求めて送れば、同ノードが a, b のどちらか一方だけを中継するよりも効率よく伝送できる。すなわち、受信ノード t_1, t_2 の両方で、a と b の２ビットを復元できる。この例では、エッジは１ビット伝送できるとしているが、ベクトル（ビット列）を伝送すると考えることができる。この場合、受信側では２つの受信ベクトルから、２つの送信ベクトルを復元する。

このようなネットワーク符号化が行われる環境においても、通信路で誤りは発生するため、誤り訂正符号に関する研究がなされている。もちろん、各ノード間の通信それぞれ (各エッジ) で誤り訂正符号化を行う問題は従来と同じである。しかし、より上位の層での誤り訂正では異なる。ソースノードで符号化し、シンクノードで復号する場合は、中継ノードで演算が行われるため、受信ベクトルに含まれる誤りに独立性がないという問題が生じる。

申請者は、この問題に関して、最尤復号を含めて、いくつかの復号法を考案した。いずれの復号法においても、復号した語のうちの一つでも誤りを含んでいれば復号誤りと定義する。最も単純な復号法では、受信語を独立に限界距離復号する。独立でない場合に、復号誤り確率と結合重み分布 (Joint Weight Distribution) の関係を明らかにした。一般には３つ以上の受信語を考えることになるため、３つ以上の符号に関する結合重み分布についても，マックウィリアムズの等式を拡張した。

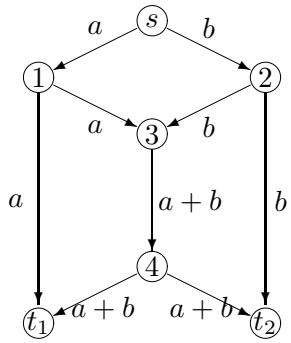

図１　バタフライネットワーク

２．研究の目的

本研究では、申請者によるこれまでの成果を発展させ、ネットワーク符号化における誤り訂正符号の復号法の性能解析手法の確立を行い、既知の符号等の性能を明らかにすることを目的とする。より詳細な目的は、以下の (1)〜(4) である。

(1) 結合重み分布計算アルゴリズムの効率的な実現: 結合重み分布に関するマックウィリアムズの等式についてさらに精査し、より使いやすい形の式の導出を目指す。また、この等式を利用し、結合重み分布の計算アルゴリズムを作成し、復号誤り確率の計算アルゴリズムを整備する。また、２元符号だけでなく、多元符号の場合についても、効率のよい実現方法を設計する。

(2) 結合重み分布計算アルゴリズムを用いての性能評価: ハミング符号や２元原始 BCH 符号、リード・マラー符号などの符号について、結合重み分布や復号誤り確率の評価を行う。その際、符号のクラスに特化した結合重み分布の性質やそれを用いた計算方法の工夫についても検討し、符号のクラス毎に効率のよい計算アルゴリズムを実現する。

(3) 各種の復号法についての評価アルゴリズムの考案とそれを用いた評価: これまで申請者が考案してきた復号アルゴリズムについて、復号誤り確率の評価方法を開発する。厳密な評価ができない場合は、上界式や下界式の導出を目指す。その結果を用いて、(2) で対象とした符号について、性能評価を行う。

(4) ネットワーク符号化における誤り訂正システムの検討: 最も簡単なシステムモデルは、ソースノードで符号化し、シンクノードだけで復号を行い、各通信路の誤り率も一定とするモデルである。しかし、各ノードの能力（送信電力も含む）等を考慮

すれば、もっと複雑なモデルを考える必要がある。そのような場合には、中継ノードでも符号化・復号を行うほうが望ましい。復号・符号化を行う中継ノードの配置方法やそのときの復号誤り率の評価法を考案する。

３．研究の方法

(1) 結合重み分布計算アルゴリズムの効率的な実現: 結合重み分布に関して、マックウィリアムズの等式についてさらに精査する。現在、導出できている等式では、もとの符号の組の結合重み分布多項式と、各要素符号を双対符号にした組の結合重み分布多項式の関係になっている。これをそれぞれの結合重みについての符号語の組数で表す方法を検討する。そして、実現の容易性について検討の上、結合重み分布の計算アルゴリズムを開発し、プログラムを作成する。また、これを用いて、受信語を独立に限界距離復号した場合について、復号誤り確率の計算プログラムを整備する。また、２元符号だけでなく、多元符号の場合についても、効率のよい実現方法を検討する。

(2) 結合重み分布計算アルゴリズムを用いての性能評価: ハミング符号や 2 元原始 BCH 符号、リード・マラー符号等具体的な符号について、結合重み分布や復号誤り確率の評価を行う。符号長は 16〜128 程度を目指す。符号に特化した性質としては、記号位置置換についての不変性やトレリス構造の性質等を考える。ハミング符号について, 2 重の結合重み分布の公式が知られているので、計算結果から、３重以上について結合重み分布の性質も精査し、可能ならば、公式化も目指す。

(3) 各種の復号法についての評価アルゴリズムの考案: これまでに、最尤復号法や準最適復号法をいくつか考案している。これらの復号アルゴリズムについて、復号誤り確率の正確な値、あるいは上界式や下界式の導出を目指す。また、得られた結果に基づき、復号誤り率を求めるプログラムを作成する。評価の結果の視点から、新たな復号アルゴリズムも検討する。

(4) ネットワーク符号化における誤り訂正システムについて: 中継ノードでも符号化・復号を行うほうが望ましい状況において、復号・符号化を行う中継ノードの配置方法やそのときの復号誤り率の評価法を考案する。

４．研究成果

研究目的のそれぞれの項目について、成果を得たが、以下では、結合重み分布の計算を中心に述べる。

符号長 n の m 個の 2 元線形符号 $C_1, C_2, \ldots, C_m$ を考え、その直積の元、すなわち、m 個の符号語の組 $(\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(m)})$、ただし、$\boldsymbol{v}^{(i)} \in C_i$ を考える。$\boldsymbol{v}^{(i)} = (v_1^{(i)}, v_2^{(i)}, \ldots, v_n^{(i)})$ とする。整数 ℓ $(1 \le \ell < 2^m)$ に対して、整数 w_ℓ を

$$w_\ell = |\{j : 1 \le j \le n \text{ and } \sum_{i=1}^{m} v_j^{(i)} 2^{m-i} = \ell\}|$$

と定めるとき、$(w_1, w_2, \ldots, w_{2^m-1})$ を $(\boldsymbol{v}^{(1)}, \boldsymbol{v}^{(2)}, \ldots, \boldsymbol{v}^{(m)})$ の結合重み (あるいは、m 結合重み) と呼ぶ。w_ℓ は, ℓ の 2 進数展開が上位ビットから順に、$v_j^{(1)} v_j^{(2)} \cdots v_j^{(m)}$ に一致するような成分位置 j の個数である。$m = 2$ の場合, w_1 は $v_j^{(1)} = 0, v_j^{(2)} = 1$ となる成分位置 j の個数、w_2 は $v_j^{(1)} = 1, v_j^{(2)} = 0$ となる個数、w_3 は $v_j^{(1)} = 1, v_j^{(2)} = 1$ となる個数となる。特に断らない限り, 以下では $m = 2$ の場合を扱う。

まず、符号語の組をすべて生成し、結合重み分布を求める方法について、他の研究者の協力を得て、マルチスレッド化や GPU (Graphic Processing Unit) を用いた計算について検討を行った。その結果、マルチコア CPU（4 コア、2 スレッド) では、計算方式や書き込み局所化などの工夫も含めて、基本アルゴリズムのシングルスレッドでの実現に比べて、7.9 倍の速度向上が得られることがわかった。また, GPU については, Fermi アーキテクチャと Kepler アーキテクチャにおいて、それぞれ 2.3 倍及び 1.5 倍の速度向上を得ている。

次に符号の構造を利用した計算速度向上について述べる。符号 C_1, C_2 に対して適当な整数 a $(1 < a < n)$ を定める。そし

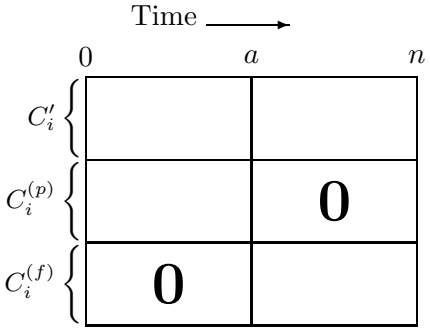

図 2　符号 C_i の生成行列の分割

て、C_i の線形部分符号 $C_i^{(p)}$, $C_i^{(f)}$ を

$$C_i^{(p)} = \{(v_1, v_2, \cdots, v_a, 0, \cdots, 0) \in C_i\}$$
$$C_i^{(f)} = \{(0, 0, \ldots, 0, v_{a+1}, \cdots, v_n) \in C_i\}$$

と定義する。

$$\begin{aligned} C_i &= C_i' + C_i^{(p)} + C_i^{(f)} \\ &= \{v_1 + v_2 + v_3 | v_1 \in C_i', \\ &\qquad v_2 \in C_i^{(p)}, v_3 \in C_i^{(f)}\} \end{aligned}$$

となる C_i の最小な線形部分符号 C_i' を考える。これらの線形部分符号を生成行列で考えると図 2 のようになる。

線形符号（あるいは線形符号のコセット）の組 C_1, C_2 において結合重み (j_1, j_2, j_3) の符号語の組の個数を A_{j_1,j_2,j_3} とするとき、

$$\mathcal{J}[C_1, C_2] = \sum_{(j_1,j_2,j_3)} A_{j_1,j_2,j_3} y_0^{n-j_1-j_2-j_3} y_1^{j_1} y_2^{j_2} y_3^{j_3}$$

を結合重み分布多項式と呼ぶ。これは、y_0 から y_3 を変数とする多項式であり、変数を明示する場合は、$\mathcal{J}_{C_1,C_2}(y_0, y_1, y_2, y_3)$ で表す。このとき、

$$\begin{aligned} &W[C_1, C_2] \\ &= \sum_{\boldsymbol{v}_1' \in C_1'} \sum_{\boldsymbol{v}_p \in C_1^{(p)}} \sum_{\boldsymbol{v}_f \in C_1^{(f)}} \\ &\quad \sum_{v_2 \in C_2'} W[(\{\boldsymbol{v}_1' + \boldsymbol{v}_p + \boldsymbol{v}_f\}, \\ &\qquad \{\boldsymbol{v}_2'\} + C_2^{(p)} + C_2^{(f)}] \end{aligned}$$

が成り立つ。$\boldsymbol{v} = (v_1, \ldots, v_n)$ の部分ベクトルを

$$p_L(\boldsymbol{v}) = (v_1, v_2, \cdots, v_i)$$
$$p_R(\boldsymbol{v}) = (v_1, v_2, \cdots, v_i)$$

と定義し、ベクトルの集合 D に対して、

$$p_L(D) = \{p_L(\boldsymbol{v}) | \boldsymbol{v} \in D\}$$
$$p_R(D) = \{p_R(\boldsymbol{v}) | \boldsymbol{v} \in D\}$$

と定義する。そして、

$$C_i^{(L)} = p_L(C_i^{(p)})$$
$$C_i^{(R)} = p_R(C_i^{(f)})$$

と定義する。

$$\begin{aligned} &W[(\{\boldsymbol{v}_1' + \boldsymbol{v}_p + \boldsymbol{v}_f\}, \{\boldsymbol{v}_2'\} + C_2^{(p)} + C_2^{(f)}] \\ &= W[p_L(\{\boldsymbol{v}_1' + \boldsymbol{v}_p\}), p_L(\{\boldsymbol{v}_1'\} + C_2^{(p)})] \cdot \\ &\quad W[(p_R(\{\boldsymbol{v}_1' + \boldsymbol{v}_f\}), p_R(\{\boldsymbol{v}_2'\} + C_2^{(f)})] \end{aligned}$$

が成り立つ。

符号が 3 個以上の結合重みの計算でも、これを利用して計算量を減らすことができる。

これらの工夫により、単一スレッドのプログラムでは, 符号長 64, 2 次のリード・マラー符号 ((64, 22) 符号) の結合重み分布を約 15 分で求めることが可能となった。これは, 100 倍以上の高速化である。したがって、マルチスレッド化、あるいは GPU 向けのプログラムでさらに高速化されることが期待できる。

2 結合重み分布多項式に関して、マックウィリアムズの等式として知られている次の関係がある。

$$\begin{aligned} &\mathcal{J}_{C_1,C_2}(y_0, y_1, y_2, y_3) \\ &= \frac{1}{|C_1^\perp|} \mathcal{J}_{C_1^\perp,C_2}(y_0 + y_2, y_1 + y_3, \\ &\qquad y_0 - y_2, y_1 - y_3) \\ &\mathcal{J}_{C_1,C_2}(y_0, y_1, y_2, y_3) \\ &= \frac{1}{|C_2^\perp|} \mathcal{J}_{C_1,C_2^\perp}(y_0 + y_1, y_0 - y_1, \\ &\qquad y_2 + y_3, y_2 - y_3) \end{aligned}$$

符号の組 C_1, C_2 の結合重み (j_1, j_2, j_3) の符号語の組の個数を A_{j_1,j_2,j_3}, 符号の組 C_1, $C_2^\perp$ の結合重み (j_1, j_2, j_3) の符号語の組の個数を B_{j_1,j_2,j_3} とする。このとき、次式が成り立つ。

$$\begin{aligned} &A_{i_1,i_2,i_3} \\ &= \frac{1}{|C_2^\perp|} \sum_{j_1=0}^{n-i_2-i_3} \sum_{j_3=0}^{i_2+i_3} B_{j_1,i_2+i_3-j_3,j_3} \\ &\quad P_{i_1}(j_1; n - i_2 - i_3) P_{i_3}(j_3; i_2 + i_3) \end{aligned}$$

が成り立つ。ここで,

$$(x+y)^{n-i}(x-y)^i = \sum_{j=0}^{n} P_k(i, n) x^{n-k} y^k$$

であり,

$$P_k(z, n) = \sum_{j=1}^{k} (-1)^j \binom{z}{j} \binom{n-z}{k-j}$$

となる。

同様に、符号の組 $C_1^\perp$, C_2 の結合重み (j_1, j_2, j_3) の符号語の組の個数を B'_{j_1,j_2,j_3} とすれば, 次式が成り立つ。

$$\begin{aligned} &A_{i_1,i_2,i_3} \\ &= \frac{1}{|C_1^\perp|} \sum_{j_2=0}^{n-i_1-i_3} \sum_{j_3=0}^{i_1+i_3} B'_{i_1+i_3-j_3,j_2,j_3} \\ &\quad P_{i_2}(j_2; n - i_1 - i_3) P_{i_3}(j_3; i_1 + i_3) \end{aligned}$$

が成り立つ。

また、次の式も知られている。

$$\begin{aligned}&\mathcal{J}_{C_1,C_2}(y_0,y_1,y_2,y_3)\\&=\frac{1}{|C_1^\perp||C_2^\perp|}\mathcal{J}_{C_1^\perp,C_2^\perp}(y_0+y_1+y_2+y_3,\\&\quad y_0-y_1+y_2-y_3,y_0+y_1-y_2-y_3,\\&\quad y_0-y_1-y_2+y_3)\end{aligned}$$

この式から、$\mathcal{J}_{C_1,C_2}(\cdot)$ を $\mathcal{J}_{C_1^\perp,C_2^\perp}(\cdot)$ から求める方法についても検討した。実現面では、個々の符号について双対符号を考える方がよいことがわかった。

$m>2$ の場合、次式が成り立つ。

$$\begin{aligned}&\mathcal{J}_{C_1,C_2,\ldots,C_\ell,\ldots,C_m}(y_0,\ldots,y_{2^m-1})\\&=\frac{1}{|C_\ell^\perp|}\mathcal{J}_{C_1,C_2,\ldots,C_\ell^\perp,\ldots,C_m}(z_0,\ldots,z_{2^m-1})\end{aligned}$$

ここで、

$$z_i=s_{m,\ell}(i)y_i+y_{i+s_{m,\ell}(i)2^{m-\ell}}$$

であり、$s_{m,\ell}(i)$ は次のように定義される。$0\le i<2^m$, $1\le\ell\le m$ について、$\varphi_{m,\ell}(i)$ を i の 2 進数展開 (m ビット) の上から ℓ ビット目とする。最上位ビットが 1 ビット目で、最下位ビットが m ビット目である。このとき、$s_{m,\ell}(i)=(-1)^{\varphi_{m,\ell}(i)}=1-2\varphi_{m,\ell}(i)$ である。$m=3$ の場合、$\varphi_{m,\ell}(i)$, $s_{m,\ell}(i)$ は以下の表のようになる。

i	$\varphi_{m,1}(i)$	$\varphi_{m,2}(i)$	$\varphi_{m,3}(i)$
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

i	$s_{m,1}(i)$	$s_{m,2}(i)$	$s_{m,3}(i)$
0	1	1	1
1	1	1	-1
2	1	-1	1
3	1	-1	-1
4	-1	1	1
5	-1	1	-1
6	-1	-1	1
7	-1	-1	-1

したがって、$m=3$ の場合、次式が成り立つ。

$$\begin{aligned}&\mathcal{J}_{C_1,C_2,C_3}(y_0,y_1,y_2,y_3,y_4,y_5,y_6,y_7)\\&=\frac{1}{|C_1^\perp|}\mathcal{J}_{C_1^\perp,C_2,C_3}(y_0+y_4,y_1+y_5,\\&\quad y_2+y_6,y_3+y_7,y_0-y_4,\\&\quad y_1-y_5,y_2-y_6,y_3-y_7)\end{aligned}$$

$$\begin{aligned}&\mathcal{J}_{C_1,C_2,C_3}(y_0,y_1,y_2,y_3,y_4,y_5,y_6,y_7)\\&=\frac{1}{|C_2^\perp|}\mathcal{J}_{C_1,C_2^\perp,C_3}(y_0+y_2,y_1+y_3,\\&\quad y_0-y_2,y_1-y_3,y_4+y_6,\\&\quad y_5+y_7,y_4-y_6,y_5-y_7)\end{aligned}$$

$$\begin{aligned}&\mathcal{J}_{C_1,C_2,C_3}(y_0,y_1,y_2,y_3,y_4,y_5,y_6,y_7)\\&=\frac{1}{|C_3^\perp|}\mathcal{J}_{C_1,C_2,C_3^\perp}(y_0+y_1,y_0-y_1,\\&\quad y_2+y_3,y_2-y_3,y_4+y_5,\\&\quad y_4-y_5,y_6+y_7,y_6-y_7)\end{aligned}$$

符号の組 C_1, C_2, C_3 の結合重み $(j_1,j_2,\ldots,j_7)$ の符号語の組の個数を $A_{j_1,j_2,\ldots,j_7}$ とし、符号の組 $C_1^\perp$, C_2, C_3 の結合重み $(j_1,j_2,\ldots,j_7)$ の符号語の組の個数を $B^{(1)}_{j_1,j_2,\ldots,j_7}$ とすれば、次式が成り立つ。

$$\begin{aligned}&A_{i_1,i_2,\ldots,i_7}\\&=\frac{1}{|C_1^\perp|}\sum_{j_4=0}^{n-i_1-i_5-i_2-i_6-i_3-i_7}\\&\quad\sum_{j_5=0}^{i_1+i_5}\sum_{j_6=0}^{i_2+i_6}\sum_{j_7=0}^{i_3+i_7}\\&\quad B^{(1)}_{i_1+i_5-j_5,i_2+i_6-j_6,i_3+i_7-j_7,j_4,j_5,j_6,j_7}\cdot\\&\qquad P_{i_4}(j_4;n-i_1-i_5-i_2-i_6\\&\qquad\qquad -i_3-i_7)P_{i_5}(j_5;i_1+i_5)\cdot\\&\qquad P_{i_6}(j_6;i_2+i_6)P_{i_7}(j_7;i_3+i_7)\end{aligned}$$

が成り立つ。同様に、C_1 以外を双対符号にしたときも関係式を導出できる。これらを用いて、双対符号の結合重み分布から元の符号の結合重み分布を求めることができる。

表 1 に、符号長 64 の 2 次のリード・マラー符号 ($(64,22)$ 符号) の最小重みの符号語どうしの結合重み分布を示す。また、この符号の結合重み分布から、3 次のリード・マラー符号 ($(64,42)$ 符号) の結合重み分布を求めた。その最小重みの符号語どうしの結合重み分布を表 2 に示す。

表 1　符号長 64 の 2 次のリード・マラー符号 ((64, 22) 符号) の最小重みの符号語どうしの結合重み分布 ($m = 2$)

i_1	i_2	i_3	A_{i_1,i_2,i_3}
0	0	16	2,604
8	8	8	468,720
12	12	4	5,832,960
16	16	0	476,532

表 2　符号長 64 の 3 次のリード・マラー符号 ((64, 42) 符号) の最小重みの符号語どうしの結合重み分布 ($m = 2$)

i_1	i_2	i_3	A_{i_1,i_2,i_3}
8	8	0	41,637,960
7	7	1	45,711,360
6	6	2	34,997,760
4	4	4	2,187,360
0	0	8	11,160

５．主な発表論文等

（研究代表者、研究分担者及び連携研究者には下線）

〔雑誌論文〕（計 1 件）

① 安藤 翔平、伊野 文彦、藤原 融、萩原 兼一、結合重み分布を拘束に計算するための並列手法、電子情報通信学会論文誌、査読有、J97-D 巻、2014、1471–80

〔学会発表〕（計 2 件）

① Shohei Ando, Fumihiko Ino, Toru Fujiwara, and Kenichi Hagihara, A Parallel Algorithm for Enumerating Joint Weight of a Binary Linear Code in Network Coding, Proceedings of the 2nd International Symposium on Networking and Computing, 査読有, 2014 pp. 137–143 Japan, (2014-12).

② 安藤 翔平、伊野 文彦、藤原 融、萩原 兼一、GPU による高速な結合重み分布生成の検討、第 13 回ハイパフォーマンスコンピューティングと計算科学シンポジウム論文集、査読無、2013、80

〔図書〕（計 0 件）

〔産業財産権〕

○出願状況（計 0 件）

○取得状況（計 0 件）

〔その他〕

なし

６．研究組織

(1) 研究代表者

藤原　融（FUJIWARA TORU）

大阪大学・大学院情報科学研究科・教授

研究者番号：70190098

(2) 研究分担者

なし

(3) 連携研究者

なし